Using Counterfactuals in Knowledge-Based Programming

Joseph Y. Halpern*
Cornell University
Dept. of Computer Science
Ithaca, NY 14853
halpern@cs.cornell.edu
http://www.cs.cornell.edu/home/halpern

Yoram Moses
Department of Electrical Engineering
Technion—Israel Institute of Technology
32000 Haifa, Israel
moses@ee.technion.ac.il

April 19, 2004

Abstract

This paper adds counterfactuals to the framework of *knowledge-based programs* of Fagin, Halpern, Moses, and Vardi [1995, 1997]. The use of counterfactuals is illustrated by designing a protocol in which an agent stops sending messages once it knows that it is safe to do so. Such behavior is difficult to capture in the original framework because it involves reasoning about counterfactual executions, including ones that are not consistent with the protocol. Attempts to formalize these notions without counterfactuals are shown to lead to rather counterintuitive behavior.

*The work was supported in part by NSF under grant IRI-96-25901 and IIS–0090145, by the Air Force Office of Scientific Research under grant F49620-96-1-0323, and by ONR under grants N00014-00-1-03-41 and N00014-01-1-0795. A preliminary version of this paper appeared in the Proceedings of the Seventh Conference on Theoretical Aspects of Rationality and Knowledge (TARK), 1998.

1 Introduction

Knowledge-based programs, first introduced by Halpern and Fagin [1989] and further developed by Fagin, Halpern, Moses, and Vardi [1995, 1997], are intended to provide a high-level framework for the design and specification of protocols. The idea is that, in knowledge-based programs, there are explicit tests for knowledge. Thus, a knowledge-based program might have the form

$$\textbf{if } K(x = 0) \textbf{ then } y := y + 1 \textbf{ else } \mathsf{skip},$$

where $K(x = 0)$ should be read as "you know $x = 0$" and skip is the action of doing nothing. We can informally view this knowledge-based program as saying "if you know that $x = 0$, then set y to $y + 1$ (otherwise do nothing)".

Knowledge-based programs are an attempt to capture the intuition that what an agent does depends on what it knows. They have been used successfully in papers such as [Dwork and Moses 1990; Hadzilacos 1987; Halpern, Moses, and Waarts 2001; Halpern and Zuck 1992; Mazer and Lochovsky 1990; Mazer 1990; Moses and Tuttle 1988; Neiger and Toueg 1993] both to help in the design of new protocols and to clarify the understanding of existing protocols. However, as we show here, there are cases when, used naively, knowledge-based programs exhibit some quite counterintuitive behavior. We then show how this can be overcome by the use of *counterfactuals* [Lewis 1973; Stalnaker 1968]. In this introduction, we discuss these issues informally, leaving the formal details to later sections of the paper.

Some counterintuitive aspects of knowledge-based programs can be understood by considering the *bit-transmission problem* from [Fagin, Halpern, Moses, and Vardi 1995]. In this problem, there are two processes, a *sender* S and a *receiver* R, that communicate over a communication line. The sender starts with one bit (either 0 or 1) that it wants to communicate to the receiver. The communication line may be faulty and lose messages in either direction in any given round. That is, there is no guarantee that a message sent by either S or R will be received. Because of the uncertainty regarding possible message loss, S sends the bit to R in every round, until S receives an *ack* message from R acknowledging receipt of the bit. R starts sending the *ack* message in the round after it receives the bit, and continues to send it repeatedly from then on. The sender S can be viewed as running the program BT_S:

$$\textbf{if } \mathit{recack} \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit},$$

where *recack* is a proposition that is true if S has already received an *ack* message from R and false otherwise, while $\mathsf{sendbit}$ is the action of sending the bit.[1] Note that BT_S is a *standard* program—it does not have tests for knowledge. We can capture some of the intuitions behind this program by using knowledge. The sender S keeps sending the bit

[1] Running such a program amounts to performing the statement repeatedly forever.

until an acknowledgment is received from the receiver R. Thus, another way to describe the sender's behavior is to say that S keeps sending the bit until it *knows* that the bit was received by R. This behavior can be characterized by the knowledge-based program BT'_S:

$$\textbf{if } K_S(\mathit{recbit}) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit},$$

where *recbit* is a proposition that is true once R has received the bit. The advantage of this program over the standard program BT_S is that it abstracts away the mechanism by which S learns that the bit was received by R. For example, if messages from S to R are guaranteed to be delivered in the same round in which they are sent, then S knows that R received the bit even if S does not receive an acknowledgment.

We might hope to improve this even further. Consider a system where all messages sent are guaranteed to be delivered, but rather than arriving in one round, they spend exactly five rounds in transit. In such a system, a sender using BT_S will send the bit 10 times, because it will take 10 rounds to get the receiver's acknowledgment after the original message is sent. The program BT'_S is somewhat better; using it S sends the bit only five times, since after the fifth round, S will know that R got his first message. Nevertheless, this seems wasteful. Given that messages are guaranteed to be delivered, it clearly suffices for the sender to send the bit once. Intuitively, the sender should be able to stop sending the message as soon as it knows that the receiver will *eventually* receive a copy of the message; the sender should not have to wait until the receiver *actually* receives it.

It seems that there should be no problem handling this using knowledge-based programs. Let $\Diamond$ be the standard "eventually" operator from temporal logic [Manna and Pnueli 1992]; $\Diamond\varphi$ means that φ is eventually true, and let $\Box$ be its dual, "always". Now the following knowledge-based program BT^*_S for the sender should capture exactly what is required:

$$\textbf{if } K_S(\Diamond \mathit{recbit}) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

Unfortunately, BT^*_S does not capture our intuitions here. To understand why, consider the sender S. Should it send the bit in the first round? According to BT^*_S, the sender S should send the bit if S does not know that R will eventually receive the bit. But if S sends the bit, then S knows that R will eventually receive it (since messages are guaranteed to be delivered in 5 rounds). Thus, S should not send the bit. Similar arguments show that S should not send the bit at any round. On the other hand, if S never sends the bit, then R will never receive it and thus S *should* send the bit! It follows that according to BT^*_S, S should send the bit exactly if it will never send the bit. Obviously, there is no way S can follow such a program. Put another way, this program cannot be implemented by a standard program at all. This is certainly not the behavior we would intuitively have expected of BT^*_S.[2]

[2] While intuitions may, of course, vary, some evidence of the counterintuitive behavior of this program is that it was used in a draft of [Fagin, Halpern, Moses, and Vardi 1995]; it was several months before we realized its problematic nature.

One approach to dealing with this problem is to change the semantics of knowledge-based programs. Inherent in the semantics of knowledge-based programs is the fact that an agent knows what standard protocol she is following. Thus, if the sender is guaranteed to send a message in round two, then she knows at time one that the message will be sent in the following round. Moreover, if communication is reliable, she also knows the message will later be received. If we weaken the semantics of knowledge sufficiently, then this problem disappears. (See [Engelhardt, van der Meyden, and Moses 1998] for an approach to dealing with the problem addressed in this paper along these lines.) However, it is not yet clear how to make this change and still maintain the attractive features of knowledge-based programs that we discussed earlier.

In this paper we consider another approach to dealing with the problem, based on counterfactuals. Our claim is that the program BT_S^* does not adequately capture our intuitions. Rather than saying that S should stop sending if S knows that R will eventually receive the bit we should, instead, say that S should stop sending if it knows that *even if S does not send another message* R will eventually receive the bit.

How should we capture this? Let $do(i, \mathsf{a})$ be the formula that is true at a point (r, m) if process i performs a in the next round.[3] The most obvious way to capture "(even) if S does not send a message then R will eventually receive the bit" uses standard implication, also known as *material implication* or *material conditional* in philosophical logic: $do(S, \mathsf{skip}) \Rightarrow recbit$. This leads to a program such as $\mathsf{BT}_S^{\Rightarrow}$:

$$\textbf{if } K_S(do(S,\mathsf{skip}) \Rightarrow \Diamond recbit) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

Unfortunately, this program does not solve our problems. It, too is not implementable by a standard program. To see why, suppose that there is some point in the execution of this protocol where S sends a message. At this point S knows it is sending a message, so S knows that $do(S, \mathsf{skip})$ is false. Thus, S knows that $do(S, \mathsf{skip}) \Rightarrow \Diamond recbit$ holds. As a result, $K_S(do(S, \mathsf{skip}) \Rightarrow \Diamond recbit)$ is true, so that the test in $\mathsf{BT}_S^{\Rightarrow}$ succeeds. Thus, according to $\mathsf{BT}_S^{\Rightarrow}$, the sender S should *not* send a message at this point. On the other hand, if S *never* sends a message according to the protocol (under any circumstance), then S knows that it will never send a message (since, after all, S knows how the protocol works). But in this case, S knows that the receiver will never receive the bit, so the test fails. Thus, according to $\mathsf{BT}_S^{\Rightarrow}$, the sender S should send the message as its first action, this time contradicting the assumption that the message is never sent. Nothing that S can do is consistent with this program.

The problem here is the use of material implication ($\Rightarrow$). Our intuitions are better captured by using counterfactual implication, which we denote by $>$. A statement such as $\varphi > \psi$ is read "if φ then ψ", just like $\varphi \Rightarrow \psi$. However, the semantics of $>$ is very different from that of $\Rightarrow$. The idea, which goes back to Stalnaker [1968] and Lewis [1973] is that a statement such as $\varphi > \psi$ is true at a world w if in the worlds "closest to" or

[3] We assume that round m takes place between time $m-1$ and m. Thus, the next round after (r, m) is round $m+1$, which takes takes place between (r, m) and $(r, m+1)$.

"most like" w where φ is true, ψ is also true. This attempts to capture the intuition that the counterfactual statement $\varphi > \psi$ stands for "if φ were the case, then ψ would hold". For example, suppose that we have a wet match and we make a statement such as "if the match were dry then it would light". Using $\Rightarrow$ this statement is trivially true, since the antecedent is false. However, with $>$, the situation is not so obvious. We must consider the worlds most like the actual world where the match is in fact dry and decide whether it would light in those worlds. If we think the match is defective for some reason, then even if it were dry, it would not light.

A central issue in the application of counterfactual reasoning to a concrete problem is that we need to specify what the "closest worlds" are. The philosophical literature does not give us any guidance on this point. We present some general approaches for doing so, motivated by our interest in modeling counterfactual reasoning about what would happen if an agent were to deviate from the protocol it is following. We believe that this example can inform similar applications of counterfactual reasoning in other contexts.

There is a subtle technical point that needs to be addressed in order to use counterfactuals in knowledge-based programs. Traditionally, we talk about a knowledge-based program Pg_{kb} being implemented by a protocol P. This is the case when the behavior prescribed by P is in accordance with what Pg_{kb} specifies. To determine whether P implements Pg_{kb}, the knowledge tests (tests for the truth of formulas of the form $K_i\varphi$) in Pg_{kb} are evaluated with respect to the points appearing in the set of runs of P. In this system, all the agents know that the properties of P (e.g. facts like process 1 always sending an acknowledgment after receiving a message from process 2) hold in all runs. But this set of runs does not account for what may happen if (counter to fact) some agents were to deviate from P. In counterfactual reasoning, we need to evaluate formulas with respect to a larger set of runs that allows for such deviations.

We deal with this problem by evaluating counterfactuals with respect to a system consisting of all possible runs (not just the ones generated by P). While working with this larger system enables us to reason about counterfactuals, processes no longer know the properties of P in this system, since it includes many runs not in P. In order to deal with this, we add a notion of likelihood to the system using what are called *ranking functions* [Spohn 1988]. Runs generated by P get rank 0; all other runs get higher rank. (Lower ranks imply greater likelihood.) Ranks let us define a standard notion of *belief.* Although a process does not *know* that the properties of P hold, it *believes* that they do. Moreover, when restricted to the set of runs of the original protocol P, this notion of belief satisfies the knowledge axiom $B_i\varphi \Rightarrow \varphi$, and coincides with the notion of knowledge we had in the original system. Thus, when the original protocol is followed, our notion of belief acts essentially like knowledge.

Using the counterfactual operator and this interpretation for belief, we get the program $\mathsf{BT}_S^>$:

$$\textbf{if } B_S(do(S, \mathsf{skip}) > \Diamond recbit) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

We show that using counterfactuals in this way has the desired effect here. If message delivery is guaranteed, then after the message has been sent once, under what seems to be the most reasonable interpretation of “the closest world” where the message is not sent, the sender believes that the bit will eventually be received. In particular, in contexts where messages are delivered in five rounds, using $\mathsf{BT}_S^{>}$, the sender will send one message.

As we said, one advantage of BT'_S over the standard program BT_S is that it abstracts away the mechanism by which S learns that the bit was received by R. We can abstract even further. The reason that S keeps sending the bit to R is that S wants R to know the value of the bit. Thus, intuitively, S should keep sending the bit until it knows that R knows its value. Let $K_R(\mathit{bit})$ be an abbreviation for $K_R(\mathit{bit} = 0) \vee K_R(\mathit{bit} = 1)$, so $K_R(\mathit{bit})$ is true precisely if R knows the value of the bit. The sender's behavior can be characterized by the following knowledge-based program, BT_S^{K}:

$$\textbf{if } K_S K_R(\mathit{bit}) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

Clearly when a message stating the value of the bit reaches the receiver, $K_R(\mathit{bit})$ holds. But it also holds in other circumstances. If, for example, the $K_S K_R(\mathit{bit})$ holds initially, then there is no need to send anything.

As above, it seems more efficient for the sender to stop sending when he knows that the receiver will *eventually* know the value of the bit. This suggests using the following program:

$$\textbf{if } K_S(do(S, \mathsf{skip}) \Rightarrow \Diamond K_R(\mathit{bit})) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

However, the same reasoning as in the case of $\mathsf{BT}^{>}$ shows that this program is not implementable. And, again, using belief and counterfactuals, we can get a program $\mathsf{BT}_S^{\Diamond B}$ that does work, and uses fewer messages than $\mathsf{BT}_S^{>}$. In fact, the following program does the job:

$$\textbf{if } B_S(do(S, \mathsf{skip}) > \Diamond B_R(\mathit{bit})) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit},$$

except that now we have to take $\mathcal{B}_R(\mathit{bit})$ to be an abbreviation for $(\mathit{bit} = 0 \wedge B_R(\mathit{bit} = 0)) \vee (\mathit{bit} = 1 \wedge B_R(\mathit{bit} = 1))$. Note that $K_R(\mathit{bit})$, which was defined to be $K_R(\mathit{bit} = 0)) \vee K_R(\mathit{bit} = 1))$, is logically equivalent to $(\mathit{bit} = 0 \wedge K_R(\mathit{bit} = 0)) \vee (\mathit{bit} = 1 \wedge K_R(\mathit{bit} = 1))$, since $K_R\varphi \Rightarrow \varphi$ is valid for any formula φ. But, in general, $B_R\varphi \Rightarrow \varphi$ is not valid, so adding the additional conjuncts in the case of belief makes what turns out to be quite an important difference. Intuitively, $B_R(\mathit{bit})$ says that R has correct beliefs about the value of the bit.

The rest of this paper is organized as follows: In the next section, there is an informal review of the semantics of knowledge-based programs. Section 3 extends the knowledge-based framework by adding counterfactuals and beliefs. We then formally analyze the programs $\mathsf{BT}_S^{>}$ and $\mathsf{BT}_S^{\Diamond B}$, showing that they have the appropropriate properties. We conclude in Section 4.

2 Giving semantics to knowledge-based programs

Formal semantics for knowledge-based programs are provided by Fagin, Halpern, Moses, and Vardi [1995, 1997]. To keep the discussion in this paper at an informal level, we simplify things somewhat here, and review what we hope will be just enough of the details so that the reader will be able to follow the main points. All the definitions in this section, except that of *de facto implementation* at the end of the section, are taken from [Fagin, Halpern, Moses, and Vardi 1995].

Informally, we view a multi-agent system as consisting of a number of interacting agents. We assume that, at any given point in time, each agent in the system is in some *local state*. A *global state* is just a tuple consisting of each agent's local state, together with the state of the *environment*, where the environment's state accounts for everything that is relevant to the system that is not contained in the state of the processes. The agents' local states typically change over time, as a result of actions that they perform. A *run* is a function from time to global states. Intuitively, a run is a complete description of what happens over time in one possible execution of the system. A *point* is a pair (r, m) consisting of a run r and a time m. If $r(m) = (\ell_e, \ell_1, \ldots, \ell_n)$, then we use $r_i(m)$ to denote process i's local state ℓ_i at the point (r, m), $i = 1, \ldots, n$ and $r_e(m)$ to denote the environment's state ℓ_e. For simplicity, time here is taken to range over the natural numbers rather than the reals (so that time is viewed as discrete, rather than dense or continuous). *Round* m in run r occurs between time $m - 1$ and m. A *system* $\mathcal{R}$ is a set of runs; intuitively, these runs describe all the possible executions of the system. For example, in a poker game, the runs could describe all the possible deals and bidding sequences.

Of major interest in this paper are the systems that we can associate with a program. To do this, we must first associate a system with a *joint protocol*. A *protocol* is a function from local states to nonempty sets of actions. (We often consider *deterministic* protocols, in which a local state is mapped to a singleton set of actions. Such protocols can be viewed as functions from local states to actions.) A joint protocol is just a set of protocols, one for each process/agent.

We would like to be able to generate the system corresponding to a given joint protocol P. To do this, we need to describe the setting, or *context*, in which P is being executed. Formally, a context γ is a tuple $(P_e, \mathcal{G}_0, \tau, \Psi)$, where P_e is a protocol for the environment, $\mathcal{G}_0$ is a set of initial global states, τ is a *transition function*, and Ψ is a set of *admissible* runs. The environment is viewed as running a protocol just like the agents; its protocol is used to capture features of the setting such as "all messages are delivered within 5 rounds" or "messages may be lost". The transition function τ describes how the actions performed by the agents and the environment change the global state by associating with each *joint action* (a tuple consisting of an action for the environment and one for each of the agents) a *global state transformer*, that is, a mapping from global states to global states. For the simple programs considered in this paper, the transition function will be almost immediate from the description of the global states. The set Ψ of admissible runs is useful for capturing various fairness properties of the context. Typically, when no fairness constraints are imposed, Ψ is the set of all runs. (For a discussion of the role of the set Ψ of admissible runs see [Fagin, Halpern, Moses, and Vardi 1995].) Since our focus in this paper is reasoning about actions and when they are performed, we assume that all contexts are such that the environment's state at the point (r, m) records the joint action performed in the previous round (that is, between $(r, m-1)$ and (r, m)). (Thus, we are essentially considering what are called *recording contexts* in [Fagin, Halpern, Moses, and Vardi 1995].)

A run r is consistent with a protocol P if it could have been generated when running protocol P. Formally, run r is *consistent with joint protocol P in context* γ if $r \in \Psi$ (so r is admissible according to the context γ), its initial global state $r(0)$ is one of the initial global states $\mathcal{G}_0$ given in γ, and for all m, the transition from global state $r(m)$ to $r(m+1)$ is the result of performing one of the joint actions specified by P and the environment protocol P_e (given in γ) in the global state $r(m)$. That is, if $P = (P_1, \ldots, P_n)$, P_e is the environment's protocol in context γ, and $r(m) = (\ell_e, \ell_1, \ldots, \ell_n)$, then there must be a joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$ such that $\mathsf{a}_e \in P_e(\ell_e)$, $\mathsf{a}_i \in P_i(\ell_i)$ for $i = 1, \ldots, n$, and $r(m+1) = \tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(r(m))$ (so that $r(m+1)$ is the result of applying the joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$ to $r(m)$. For future reference, we will say that a run r is *consistent with* γ if r is consistent with *some* joint protocol P in γ. A system $\mathcal{R}$ *represents* a joint protocol P in a context γ if it consists of all runs in Ψ consistent with P in γ. We use $\mathbf{R}(P, \gamma)$ to denote the system representing P in context γ.

The basic logical language $\mathcal{L}$ that we use is a standard propositional temporal logic. We start out with a set Φ of primitive propositions $p, q, \ldots$ (which are sometimes given more meaningful names such as *recbit* or *recack*). Every primitive proposition is considered to be a formula of $\mathcal{L}$. We close off under the Boolean operators $\wedge$ (conjunction) and $\neg$ (negation). Thus, if φ and ψ are formulas of $\mathcal{L}$, then so are $\neg\varphi$ and $\varphi \wedge \psi$. The other Boolean operators are definable in terms of these. E.g., implication $\varphi \Rightarrow \psi$ is defined as $\neg(\neg\varphi \wedge \psi)$. Finally, we close off under temporal operators. For the purposes of this paper, it suffices to consider the standard linear-time temporal operators $\bigcirc$ ("in the next (global) state')' and $\Diamond$ ("eventually"): If φ is a formula, then so are $\bigcirc\varphi$ and $\Diamond\varphi$. The dual of $\Diamond$, which stands for "forever," is denoted by $\Box$ and defined to be shorthand for $\neg\Diamond\neg$. This completes the definition of the language.

In order to assign meaning to the formulas of such a language $\mathcal{L}$ in a system $\mathcal{R}$, we need an *interpretation* π, which determines the truth of the primitive propositions at each of the global states of $\mathcal{R}$. Thus, $\pi : \Phi \times \mathcal{G} \to \{\mathbf{true}, \mathbf{false}\}$, where $\pi(p, g) = \mathbf{true}$ exactly if the proposition p is true at the global state g. An *interpreted system* is a pair $\mathcal{I} = (\mathcal{R}, \pi)$ where $\mathcal{R}$ is a system as before, and π is an interpretation for Φ in $\mathcal{R}$. Formulas of $\mathcal{L}$ are considered true or false at a point (r, m) with respect to an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ where $r \in \mathcal{R}$. Formally,

- $(\mathcal{I}, r, m) \models p$, for $p \in \Phi$, iff $\pi(p, r(m)) = \mathbf{true}$.

- $(\mathcal{I}, r, m) \models \neg\varphi$, iff $(\mathcal{I}, r, m) \not\models \varphi$.

- $(\mathcal{I}, r, m) \models \varphi \wedge \psi$, iff both $(\mathcal{I}, r, m) \models \varphi$ and $(\mathcal{I}, r, m) \models \psi$.
- $(\mathcal{I}, r, m) \models \bigcirc\varphi$, iff $(\mathcal{I}, r, m+1) \models \varphi$.
- $(\mathcal{I}, r, m) \models \Diamond\varphi$, iff $(\mathcal{I}, r, m') \models \varphi$ for some $m' \geq m$.

By adding an interpretation π to the context γ, we obtain an *interpreted context* (γ, π).

We now describe a simple programming language, introduced in [Fagin, Halpern, Moses, and Vardi 1995], which is still rich enough to describe protocols, and whose syntax emphasizes the fact that an agent performs actions based on the result of a test that is applied to her local state. A (*standard*) *program* for agent i is a statement of the form:

```
case of
    if $t_1$ do $\mathsf{a}_1$
    if $t_2$ do $\mathsf{a}_2$
    ...
end case
```

where the t_j's are *standard tests* for agent i and the a_j's are actions of agent i (i.e., $\mathsf{a}_j \in ACT_i$). (We later modify these programs to obtain *knowledge-based* and *belief-based* programs; the distinction will come from the kinds of tests allowed. We omit the **case** statement if there is only one clause.) A standard test for agent i is simply a propositional formula over a set Φ_i of primitive propositions. Intuitively, if L_i represents the local states of agent i in $\mathcal{G}$, then once we know how to evaluate the tests in the program at the local states in L_i, we can convert this program to a protocol over L_i: at a local state ℓ, agent i nondeterministically chooses one of the (possibly infinitely many) clauses in the **case** statement whose test is true at ℓ, and executes the corresponding action.

We want to use an interpretation π to tell us how to evaluate the tests. However, not just any interpretation will do. We intend the tests in a program for agent i to be *local*, that is, to depend only on agent i's local state. It would be inappropriate for agent i's action to depend on the truth value of a test that i could not determine from her local state. An interpretation π on the global states in $\mathcal{G}$ is *compatible* with a program Pg_i for agent i if every proposition that appears in Pg_i is local to i; that is, if q appears in Pg_i, the states s and s' are in $\mathcal{G}$, and $s \sim_i s'$, then $\pi(s)(q) = \pi(s')(q)$. If φ is a propositional formula all of whose primitive propositions are local to agent i, and ℓ is a local state of agent i, then we write $(\pi, \ell) \models \varphi$ if φ is satisfied by the truth assignment $\pi(s)$, where $s = (s_e, s_1, \ldots, s_n)$ is a global state such that $s_i = \ell$. Because all the primitive propositions in φ are local to i, it does not matter which global state s we choose, as long as i's local state in s is ℓ. Given a program Pg_i for agent i and an interpretation π compatible with Pg_i, we define a protocol that we denote Pg_i^π by setting:

$$\mathsf{Pg}_i^\pi(\ell) = \begin{cases} \{\mathsf{a}_j : (\pi, \ell) \models t_j\} & \text{if } \{j : (\pi, \ell) \models t_j\} \neq \emptyset \\ \{\mathsf{skip}\} & \text{if } \{j : (\pi, \ell) \models t_j\} = \emptyset. \end{cases}$$

Intuitively, Pg_i^π selects all actions from the clauses that satisfy the test, and selects the null action skip if no test is satisfied. In general, we get a nondeterministic protocol, since more than one test may be satisfied at a given state.

Many of the definitions that we gave for protocols have natural analogues for programs. We define a *joint* program to be a tuple $\mathsf{Pg} = (\mathsf{Pg}_1, \ldots, \mathsf{Pg}_n)$, where Pg_i is a program for agent i. An interpretation π is *compatible* with Pg if π is compatible with each of the Pg_i's. From Pg and π we get a joint protocol $\mathsf{Pg}^\pi = (\mathsf{Pg}_1^\pi, \ldots, \mathsf{Pg}_n^\pi)$. We say that an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ *represents* a joint program Pg in the interpreted context (γ, π) exactly if π is compatible with Pg and $\mathcal{I}$ represents the corresponding protocol Pg^π. We denote the interpreted system representing Pg in (γ, π) by $\mathbf{I}(\mathsf{Pg}, \gamma, \pi)$. Of course, this definition only makes sense if π is compatible with Pg. From now on we always assume that this is the case.

The syntactic form of our standard programs is in many ways more restricted than that of programs in common programming languages such as C or FORTRAN. In such languages, one typically sees constructs such as **for**, **while**, or **if...then...else...**, which do not have syntactic analogues in our formalism. As discussed in [Fagin, Halpern, Moses, and Vardi 1995], it is possible to encode a program counter in tests and actions of standard programs. By doing so, it is possible to simulate these constructs. Hence, there is essentially no loss of generality in our definition of standard programs.

Since each test in a standard program Pg run by process i can be evaluated in each local state, we can derive a protocol from Pg in an obvious way: to find out what process i does in a local state ℓ, we evaluate the tests in the program in ℓ and perform the appropriate action. A run is *consistent with* Pg *in interpreted context* (γ, π) if it is consistent with the protocol derived from Pg. Similarly, a system *represents* Pg *in interpreted context* (γ, π) if it represents the protocol derived from Pg in (γ, π).

Example 2.1 Consider the (joint) program $\mathsf{BT} = (\mathsf{BT}_S, \mathsf{BT}_R)$, where BT_S is as defined in the introduction, and BT_R is the program

$$\textbf{if } \mathit{recbit} \textbf{ then } \mathsf{sendack} \textbf{ else } \mathsf{skip}.$$

Thus, in BT_R, the receiver sends an acknowledgement if it has received the bit, and otherwise does nothing. This program, like all the programs considered in this paper, is applied repeatedly, so it effectively runs forever. Assume that S's local state includes the time, its input bit, and whether or not S has received an acknowledgment from R; the state thus has the form (m, i, x), where m is a natural number (the time), $i \in \{0, 1\}$ is the input bit, and $x \in \{\lambda, \mathit{ack}\}$. Similarly, R's local state has the form (m, x), where m is the time and x is either λ, 0, or 1, depending on whether or not it has received the bit from S and what the bit is. As in all recording contexts, the environment state keeps track of the actions performed by the agents. Since the environment state plays no role here, we omit it from the description of the global state, and just identify the global state with the pair consisting of S and R's local state. Suppose that, in context γ, the environment protocol nondeterministically decides whether or not a message sent by S and/or R is delivered, the initial global states are $((0,0,\lambda),(0,\lambda))$ and $((0,1,\lambda),(0,\lambda))$, the transition function is such that the joint actions have the obvious effect on the global state, and all runs are admissible. Then a run consistent with BT in (γ,π) in which S's bit is 0, R receives the bit in the second round, and S receives an acknowledgment from R in the third round has the following sequence of global states: $((0,0,\lambda),(0,\lambda)),((1,0,\lambda),(1,\lambda)),((2,0,\lambda),(2,0)),((3,0,\mathit{ack}),(3,0)),((4,0,\mathit{ack}),(4,0)),\ldots$. ∎

Now we consider knowledge-based programs. We start by extending our logical language by adding a modal operator K_i for every agent $i = 1,\ldots,n$. Thus, whenever φ is a formula, so is $K_i\varphi$. Let $\mathcal{L}_K$ be the resulting language. According to the standard definition of knowledge in systems [Fagin, Halpern, Moses, and Vardi 1995], an agent i knows a fact φ at a given point (r,m) in interpreted system $\mathcal{I} = (\mathcal{R},\pi)$ if φ is true at all points in $\mathcal{R}$ where i has the same local state as it does at (r,m). We now have

- $(\mathcal{I},r,m) \models K_i\varphi$ if $(\mathcal{I},r',m') \models \varphi$ for all points (r',m') such that $r_i(m) = r'_i(m')$.

Thus, i knows φ at the point (r,m) if φ holds at all points consistent with i's information at (r,m).

A *knowledge-based program* has the same structure as a standard program except that all tests in the program text Pg_i for agent i are formulas of the form $K_i\psi$.[4] As for standard programs, we can define when a protocol implements a knowledge-based program, except this time it is with respect to an interpreted context. The situation in this case is, however, somewhat more complicated. In a given context, a process can determine the truth of a standard test such as "$x = 0$" by simply checking its local state. However, the truth of the tests for knowledge that appear in knowledge-based programs cannot in general be determined simply by looking at the local state in isolation. We need to look at the whole system. As a consequence, given a run, we cannot in general determine if it is consistent with a knowledge-based program in a given interpreted context. This is because we cannot tell how the tests for knowledge turn out without being given the other possible runs of the system; what a process knows at one point will depend in general on what other points are possible. This stands in sharp contrast to the situation for standard programs.

This means it no longer makes sense to talk about a run being consistent with a knowledge-based program in a given context. However, notice that, given an interpreted system $\mathcal{I} = (\mathcal{R},\pi)$, we can derive a protocol from a knowledge-based program Pg_{kb} for process i by evaluating the knowledge tests in Pg_{kb} with respect to $\mathcal{I}$. That is, a test such as $K_i\varphi$ holds in a local state ℓ if φ holds at all points (r,m) in $\mathcal{I}$ such that

[4] All standard programs can be viewed as knowledge-based programs. Since all the tests in a standard program for agent i must be local to i, every test φ in a standard program for agent i is equivalent to $K_i\varphi$.

$r_i(m) = \ell$.[5] In general, different protocols can be derived from a given knowledge-based program, depending on what system we use to evaluate the tests. Let $\mathsf{Pg}_{kb}^{\mathcal{I}}$ denote the protocol derived from Pg_{kb} by using $\mathcal{I}$ to evaluate the tests for knowledge. An interpreted system $\mathcal{I}$ *represents* the knowledge-based program Pg_{kb} in interpreted context (γ, π) if $\mathcal{I}$ represents the protocol $\mathsf{Pg}_{kb}^{\mathcal{I}}$. That is, $\mathcal{I}$ represents Pg_{kb} if $\mathcal{I} = \mathbf{I}(\mathsf{Pg}_{kb}^{\mathcal{I}}, \gamma, \pi)$. Thus, a system represents Pg_{kb} if it satisfies a certain fixed-point equation. A protocol P *implements* Pg_{kb} in interpreted context (γ, π) if $P = \mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$.

This definition is somewhat subtle, and determining the protocol(s) implementing a given knowledge-based program may be nontrivial. Indeed, as shown by Fagin, Halpern, Moses, and Vardi [1995, 1997], in general, there may be no protocols implementing a knowledge-based program Pg_{kb} in a given context, there may be only one, or there may be more than one, since the fixed-point equation may have no solutions, one solution, or many solutions. In particular, it is not hard to show that there is no (joint) protocol implementing a (joint) program where S uses BT_S^* or $\mathsf{BT}_S^{\Rightarrow}$, as described in the introduction.

For the purposes of this paper, it is useful to have a notion slightly weaker than that of implementation. Two joint protocols $P = (P_1, \ldots, P_n)$ and $P' = (P_1', \ldots, P_n')$ are *equivalent in context* γ, denoted $P \approx_\gamma P'$, if (a) $\mathbf{R}(P, \gamma) = \mathbf{R}(P', \gamma)$ and (b) $P_i(\ell) = P_i'(\ell)$ for every local state $\ell = r_i(m)$ with $r \in \mathbf{R}(P, \gamma)$. Thus, two protocols that are equivalent in γ may disagree on the actions performed in some local states, provided that those local states never arise in the actual runs of these protocols in γ. We say P *de facto implements* a knowledge-based program Pg_{kb} in interpreted context (γ, π) if $P \approx_\gamma \mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$. Arguably, de facto implementation suffices for most purposes, since all we care about are the runs generated by the protocol. We do not care about the behavior of the protocol on local states that never arise.

It is almost immediate from the definition that if P implements Pg_{kb}, then P de facto implements Pg_{kb}. The converse may not be true, since we may have $P \approx_\gamma \mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$ without having $P = \mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$. On the other hand, as the following lemma shows, if P de facto implements Pg_{kb}, then a protocol closely related to P implements Pg_{kb}.

Lemma 2.2 *If P de facto implements Pg_{kb} in (γ, π) then $\mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$ implements Pg_{kb} in (γ, π).*

Proof Suppose that P de facto implements Pg_{kb} in (γ, π). Let $P' = \mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$. By definition, $P' \approx_\gamma P$. Thus, $\mathbf{I}(P', \gamma, \pi) = \mathbf{I}(P, \gamma, \pi)$, so $P' = \mathsf{Pg}_{kb}^{\mathbf{I}(P',\gamma,\pi)}$. It follows that P' implements Pg_{kb}. ▮

[5] Note that if there is no point (r, m) in $\mathcal{I}$ such that $r_i(m) = \ell$, then $K_i\varphi$ vacuously holds at ℓ, for all formulas φ.

3 Counterfactuals and Belief

In this section, we show how counterfactuals and belief can be added to the knowledge-based framework, and use them to do a formal analysis of the programs $\mathsf{BT}_S^{>}$ and $\mathsf{BT}_S^{\Diamond B}$ from the introduction.

3.1 Counterfactuals

The semantics we use for counterfactuals is based on the standard semantics used in the philosophy literature [Lewis 1973; Stalnaker 1968]. As with other modal logics, this semantics starts with a set W of possible worlds. For every possible world $w \in W$ there is a (partial) order $<_w$ defined on W. Intuitively, $w_1 <_w w_2$ if w_1 is "closer" or "more similar" to world w than w_2 is. This partial order is assumed to satisfy certain constraints, such as the condition that $w <_w w'$ for all $w' \neq w$: world w is closer to w than any other world is. A counterfactual statement of the form $\varphi > \psi$ is then taken to be true at a world w if, in all the worlds closest to w among the worlds where φ is true, ψ is also true.

In our setting, we obtain a notion of closeness by associating with every point (r, m) of a system $\mathcal{I}$ a partial order on the points of $\mathcal{I}$.[6] An *order assignment* for a system $\mathcal{I} = (\mathcal{R}, \pi)$ is a function $\lll$ that associates with every point (r, m) of $\mathcal{I}$ a partial order relation $\lll_{(r,m)}$ over the points of $\mathcal{I}$. The partial orders must satisfy the constraint that (r, m) is a minimal element of $\lll_{(r,m)}$, so that there is no run $r' \in \mathcal{R}$ and time $m' \geq 0$ satisfying $(r', m') \lll_{(r,m)} (r, m)$. A *counterfactual system* is a pair of the form $\mathcal{J} = (\mathcal{I}, \lll)$, where $\mathcal{I}$ is an interpreted system as before, while $\lll$ is an order assignment for the points in $\mathcal{I}$. Given a counterfactual system $\mathcal{J} = (\mathcal{I}, \lll)$, a point (r, m) in $\mathcal{I}$, and a set A of points of $\mathcal{I}$, define

$$\begin{aligned}\texttt{closest}(A, (r, m), \mathcal{J}) \quad = \\ &\{(r', m') \in A : \text{ there is no } (r'', m'') \in A \text{ such that } (r'', m'') \lll_{(r,m)} (r', m')\}.\end{aligned}$$

Thus, $\texttt{closest}(A, (r, m), \mathcal{J})$ consists of the closest points to (r, m) among the points in A (according to the order assignment $\lll$).

To allow for counterfactual statements, we extend our logical language $\mathcal{L}$ with a binary operator $>$ on formulas, so that whenever φ and ψ are formulae, so is $\varphi > \psi$. We read $\varphi > \psi$ as "*if φ were the case, then ψ*," and denote the resulting language by $\mathcal{L}^{>}$.

Let $[\![\varphi]\!] = \{(r, m) : (\mathcal{J}, r, m) \models \varphi\}$; that is, $[\![\varphi]\!]$ consists of all points in $\mathcal{J}$ satisfying φ. We can now define the semantics of counterfactuals as follows:

$$(\mathcal{J}, r, m) \models \varphi > \psi \;\text{ if }\; (\mathcal{J}, r', m') \models \psi \;\text{ for all } (r', m') \in \texttt{closest}([\![\varphi]\!], (r, m), \mathcal{J}).$$

[6] In a more general treatment, we could associate a different partial order with every agent at every point; this is not necessary for the examples we consider in this paper.

This definition captures the intuition for counterfactuals stated earlier: $\varphi > \psi$ is true at a point (r, m) if ψ is true at the points closest to (r, m) where φ is true.

All earlier analyses of (epistemic) properties of a protocol P in a context γ used the interpreted system $\mathbf{I}(P, \gamma, \pi)$, consisting of all the runs consistent with P in context γ. However, counterfactual reasoning involves events that occur on runs that are not consistent with P. To support such reasoning we need to consider runs not in $\mathbf{I}(P, \gamma, \pi)$. The runs that must be added can, in general, depend on the type of counterfactual statements allowed in the logical language. Thus, for example, if we allow formulas of the form $do(i, \mathsf{a}) > \psi$ for process i and action a, then we must allow, at every point of the system, a possible future in which i's next action is a.[7]

An even richer set of runs is needed if we allow the language to specify a sequence of actions performed by a given process, or if counterfactual conditionals $>$ can be nested. To handle a broad class of applications, including ones involving formulas with temporal operators and arbitrary nesting of conditional statements involving $do(i, \mathsf{a})$, we do reasoning with respect to the system $\mathcal{I}^+(\gamma, \pi) = (\mathcal{R}^+(\gamma), \pi)$ consisting of *all* runs compatible γ, that is, all runs consistent with some protocol P' in context γ. In this way all possible behaviors, within the constraints induced by γ, can be reasoned about. There is a potential problem with using system $\mathcal{I}^+(\gamma, \pi) = (\mathcal{R}^+(\gamma), \pi)$ for reasoning about P: all reference to P has been lost. We return to this issue in the next section, when we discuss belief. For now we show how to use $\mathcal{I}^+(\gamma, \pi)$ as a basis for doing counterfactual reasoning.

As we have already discussed, the main issue in using $\mathcal{I}^+(\gamma, \pi)$ to reason about P is that of defining an appropriate order assignment. We are interested in order assignments that depend on the protocol in a uniform way. An *order generator* o for a context γ is a function that associates with every protocol P an order assignment $\ll^P = o(P)$ on the points of $\mathcal{R}^+(\gamma)$. A *counterfactual context* is a tuple $\zeta = (\gamma, \pi, o)$, where o is an order generator for γ. In what follows we denote by $\mathcal{J}^c(P, \zeta)$ the counterfactual system $(\mathcal{I}^+(\gamma, \pi), o(P))$, where $\zeta = (\gamma, \pi, o)$; we omit ζ when it is clear from context.

We are interested in order generators o such that $o(P)$ says something about deviations from P. For the technical results we prove in the rest of the paper, we focus on order generators that prefer runs in which the agents do not deviate from their protocol. Given an agent i, action a, protocol P, context γ, and point (r, m) in $\mathcal{R}^+(\gamma)$, define $\mathtt{close}(i, \mathsf{a}, P, \gamma, (r, m)) = \{(r', m) : \text{(a) } r' \in \mathcal{R}^+(\gamma), \text{ (b) } r'(m') = r(m')$ for all $m' \leq m$, (c) if agent i performs a in round $m + 1$ of r, then $r' = r$, (d) if agent i does

[7] Recall from the introduction that our programs use the formula $do(i, \mathsf{a})$ to state that agent i is about to perform action a. Thus, $do(i, \mathsf{a}) > \varphi$ says "if agent i were to perform a then φ would be the case." We assume that all interpretations we consider give this formula the appropriate meaning. If the protocol P being used is encoded in the global state (for example, if it is part of the environment state), then we can take $do(i, \mathsf{a})$ to be a primitive proposition. Otherwise, we cannot, since its truth cannot be determined from the global state. However, we can always take $do(i, \mathsf{a})$ to be an abbreviation for $\bigcirc last(i, \mathsf{a})$, where the interpretation π ensures that $last(i, \mathsf{a})$ is true at a point (r, m) if i performed a in round m of r. Since we assume the the last joint action performed is included in the environment state, the truth of $last(i, \mathsf{a})$ is determined by the global state.

not perform perform a in round $m + 1$ of r, then agent i performs a in round $m + 1$ of r' and follows P in all later rounds, (e) all agents other than i follow P from round $m + 1$ on in $r'\}$. That is, $\texttt{close}(i, \mathsf{a}, P, \gamma, (r, m))$ is the set of points (r', m) where run $r' = r$ if i performs a in round $m + 1$ of r; otherwise, r' is identical to r up to time m and all the agents act according to joint protocol P at all later times, except that at the point (r', m), agent i performs action a. An order generator o for γ *respects protocols* if, for every protocol P, point (r, m) of $\mathbf{R}(P, \gamma)$, action a, and agent i, $\texttt{closest}([\![do(i, \mathsf{a})]\!], (r, m), \mathcal{J}^c(P))$ is a nonempty subset of $\texttt{close}(i, \mathsf{a}, P, \gamma, (r, m))$ that includes (r, m). Of course, the most obvious order generator that respects protocols just sets $\texttt{closest}([\![do(i, \mathsf{a})]\!], (r, m), \mathcal{J}^+(P)) = \texttt{close}(i, \mathsf{a}, P, \gamma, (r, m))$. Since our results hold for arbitrary order generators that respect protocols, we have allowed the extra flexibility of allowing $\texttt{closest}([\![do(i, \mathsf{a})]\!], (r, m), \mathcal{J}^+(P))$ to be a strict subset of $\texttt{close}(i, \mathsf{a}, P, \gamma, (r, m))$.

A number of points are worth noting about this definition:

- If the environment's protocol P_e and the agents' individual protocols in P are all deterministic, then $\texttt{close}(i, \mathsf{a}, P, \gamma, (r, m))$ is a singleton, since there is a unique run where the agents act according to joint protocol P at all times except that agent i performs action a at time m. Thus, $\texttt{closest}([\![do(i, \mathsf{a})]\!], (r, m), \mathcal{J}^c(P))$ must be the singleton $\texttt{close}(i, \mathsf{a}, P, \gamma, (r, m))$ in this case. However, in many cases, it is best to view the environment as following a nondeterministic protocol (for example, nondeterministically deciding at which round a message will be delivered); in this case, there may be several points in $\mathcal{I}$ closest to (r, m). Stalnaker [1968] required there to be a unique closest world; Lewis [1973] did not. There was later discussion of how reasonable this requirement was (see, for example, [Stalnaker 1980]). Thinking in terms of systems may help inform this debate.

- If process i does not perform action a at the point (r, m), then there may be points in $\texttt{closest}([\![do(i, \mathsf{a})]\!], (r, m), \mathcal{J}^c(P))$ that are not in $\mathbf{R}(P, \gamma)$, even if $r \in \mathbf{R}(P, \gamma)$. These points are "counter to fact".

- According to our definition, the notion of "closest" depends on the protocol that generates the system. For example, consider a context γ' that is just like the context γ from Example 2.1, except that S keeps track in its local state, not only of the time, but also of the number of messages it has sent. Suppose that the protocol P_S for S is determined by the program

  $$\textbf{if } \mathit{time}{=}0 \textbf{ then } \mathsf{sendbit} \textbf{ else } \mathsf{skip},$$

  while P'_S is the protocol determined by the program

  $$\textbf{if } \#\mathit{messages}{=}0 \textbf{ then } \mathsf{sendbit} \textbf{ else } \mathsf{skip}.$$

  Let $P = (P_S, \mathsf{SKIP}_R)$ and $P' = (P'_S, \mathsf{SKIP}_R)$, where P_R is the protocol where R does nothing (performs the action skip) in all states. Clearly $\mathbf{R}(P, \gamma') = \mathbf{R}(P', \gamma')$:

whether it is following P_S or P'_S, the sender S sends a message only in the first round of each run. It follows that these two protocols specify exactly the same behavior in this context. While these protocols coincide when no deviations take place, they may differ if deviations are possible. For example, imagine a situation where, for whatever reason, S did nothing in the first round. In that case, at the end of the first round, the clock has advanced from 0, while the count of the number of messages that S has sent is still 0. P and P' would then produce different behavior in the second round. This difference is captured by our definitions. If o respects protocols, then $\texttt{closest}([\![do(S,\mathsf{skip})]\!],(r,0),\mathcal{J}^c(P)) \neq \texttt{closest}([\![do(S,\mathsf{skip})]\!],(r,0),\mathcal{J}^c(P'))$. No messages are sent by S in runs appearing in points in $\texttt{closest}([\![do(S,\mathsf{skip})]\!],(r,0),\mathcal{J}^c(P))$, while exactly one message is sent by S in each run appearing in points in $\texttt{closest}([\![do(S,\mathsf{skip})]\!],(r,0),\mathcal{J}^c(P'))$.

This dependence on the protocol is a deliberate feature of our definition; by using order generators, the order assignment we consider is a function of the protocol being used. While the protocols P and P' specify the same behavior in γ, they specify different behavior in "counterfactual" runs, where something happens that somehow causes behavior inconsistent with the protocol. The subtle difference between the two protocols is captured by our definitions.

3.2 Belief

As we have just seen, in order to allow for counterfactual reasoning about a protocol P in a context γ, our model needs to represent "counterfactual" runs that do not appear in $\mathbf{R}(P,\gamma)$. Using the counterfactual system $\mathcal{J}^c(P)$, which includes all runs of $\mathcal{R}^+(\gamma)$, provides considerable flexibility and generality in counterfactual reasoning. However, doing this has a rather drastic impact on the processes' knowledge of the protocol being used. Agents have considerable knowledge of the properties of protocol P in the interpreted system $\mathbf{I}(P,\gamma)$, since it contains only the runs of $\mathbf{R}(P,\gamma)$. For example, if agent 1's first action in P is always b, then all agents are guaranteed to know this fact (provided that it is expressible in the language, of course); indeed, this fact will be *common knowledge*, which means agent knows it, for any depth of nesting of these knowledge statements (cf. [Fagin, Halpern, Moses, and Vardi 1995; Halpern and Moses 1990]). If we evaluate knowledge with respect to $\mathcal{R}^+(\gamma)$, then the agents have lost the knowledge that they are running protocol P. We deal with this by adding extra information to the models that allows us to capture the agents' beliefs. Although the agents will not *know* they are running protocol P, they will *believe* that they are.

A *ranking function* for a system $\mathcal{R}$ is a function $\kappa : \mathcal{R} \to \mathbf{N}^+$, associating with every run of $\mathcal{R}$ a *rank* $\kappa(r)$, which is either a natural number or ∞, such that $\min_{r\in\mathcal{R}} \kappa(r) = 0$.[8]

[8] The similarity in notation with the κ-rankings of [Goldszmidt and Pearl 1992], which are based on Spohn's *ordinal conditional functions* [1988], is completely intentional. Indeed, everything we are saying here can be recast in Spohn's framework.

Intuitively, the rank of a run defines the likelihood of the run. Runs of rank 0 are most likely; runs of rank 1 are somewhat less likely, those of rank 2 are even more unlikely, and so on. Very roughly speaking, if $\epsilon > 0$ is small, we can think of the runs of rank k as having probability $O(\epsilon^k)$. For our purposes, the key feature of rankings is that they can be used to define a notion of belief (cf. [Friedman and Halpern 1997]). Intuitively, of all the points considered possible by a given agent at a point (r, m), the ones believed to have occurred are the ones appearing in runs of minimal rank. More formally, for a point (r, m) define

$$\mathsf{min}_i^\kappa(r,m) \;=\; \min\{\kappa(r') \mid r' \in \mathcal{R}(\gamma) \text{ and } r'_i(m') = r_i(m) \text{ for some } m' \geq 0\}.$$

Thus, $\mathsf{min}_i^\kappa(r,m)$ is the minimal κ-rank of runs in which $r_i(m)$ appears as a local state for agent i.

An *extended system* is a triple of the form $\mathcal{J} = (\mathcal{I}, \lll, \kappa)$, where $(\mathcal{I}, \lll)$ is a counterfactual system, and κ is a ranking function for the runs of $\mathcal{I}$. In extended systems we can define a notion of belief. The logical language that results from closing $\mathcal{L}^>$ (resp. $\mathcal{L}$) under belief operators B_i, for $i = 1, \ldots, n$, is denoted $\mathcal{L}_B^>$ (resp. $\mathcal{L}_B$). The truth of $B_i\varphi$ is defined as follows:

$$\begin{array}{lll}(\mathcal{I}, \lll, \kappa, r, m) \models B_i\varphi & \text{iff} & (\mathcal{I}, \lll, \kappa, r', m') \models \varphi \text{ for all } (r', m') \text{ such that} \\ & & \kappa(r') = \mathsf{min}_i^\kappa(r,m) \text{ and } r'_i(m') = r_i(m).\end{array}$$

What distinguishes knowledge from belief is that knowledge satisfies the *knowledge axiom*: $K_i\varphi \Rightarrow \varphi$ is valid. While $B_i\varphi \Rightarrow \varphi$ is not valid, it is true in runs of rank 0.

Lemma 3.1 *Suppose that $\mathcal{J} = ((\mathcal{R}, \pi), \lll, \kappa)$ is an extended system, $r \in \mathcal{R}$, and $\kappa(r) = 0$. Then for every formula φ and all times m, we have $(\mathcal{J}, r, m) \models B_i\varphi \Rightarrow \varphi$.*

Proof Assume that $\kappa(r) = 0$. Thus, $\mathsf{min}_i^\kappa(r,m) = 0$ for all times $m \geq 0$. It now immediately follows from the definitions that if $(\mathcal{J}, r, m) \models B_i\varphi$, then $(\mathcal{J}, r, m) \models \varphi$. ∎

By analogy with order generators, we now want a uniform way of associating with each protocol P a ranking function. Intuitively, we want to do this in a way that lets us recover P. We say that a ranking function κ is *P-compatible* (for γ) if $\kappa(r) = 0$ if and only if $r \in \mathbf{R}(P, \gamma)$. A *ranking generator* for a context γ is a function σ ascribing to every protocol P a ranking $\sigma(P)$ on the runs of $\mathcal{R}^+(\gamma)$. A ranking generator σ is *deviation compatible* if $\sigma(P)$ is P-compatible for every protocol P. An obvious example of a deviation-compatible ranking generator is the *characteristic* ranking generator σ_ξ that, for a given protocol P, yields a ranking that assigns rank 0 to every run in $\mathbf{R}(P, \gamma)$ and rank 1 to all other runs. This captures the assumption that runs of P are likely and all other runs are unlikely, without attempting to distinguish among them. Another deviation-compatible ranking generator is σ^*, where the ranking $\sigma^*(P)$ assigns to a run r the total number of times that agents deviate from P in r. Obviously, $\sigma^*(P)$ assigns r

the rank 0 exactly if $r \in \mathbf{R}(P, \gamma)$, as desired. Intuitively, σ^* captures the assumption that not only are deviations unlikely, but they are independent. It is clearly possible to construct other P-compatible rankings that embody other assumptions. For example, deviation can taken to be an indication of faulty behavior. Runs of rank k can be those where exactly k processes are faulty.

Our interest in deviation-compatible ranking generators is motivated by the observation that the notion of belief that they give rise to in $\mathcal{I}^+(\gamma, \pi)$ generalizes the notion of knowledge with respcet to $\mathbf{I}(P, \gamma, \pi)$. To make this precise, define φ^B to be the formula that is obtained by replacing all K_i operators in φ by B_i. (Notice that if $\varphi \in \mathcal{L}_K$ then $\varphi^B \in \mathcal{L}_B$.) In addition, since ranking generators now play a role in determining beliefs, we define an *interpreted belief context* to be a triple of the form (γ, π, σ).

Theorem 3.2 *Let σ be a deviation-compatible ranking generator for γ. For every formula $\varphi \in \mathcal{L}_K$ and for all points (r, m) of $\mathcal{R} = \mathbf{R}(P, \gamma)$ and every ordering $\lll$ we have*

$$(\mathbf{I}(P, \gamma, \pi), r, m) \models \varphi \quad \textit{iff} \quad (\mathcal{I}^+(\gamma, \pi), \lll, \sigma(P), r, m) \models \varphi^B.$$

Proof We proceed by induction on the structure of φ. For primitive propositions, the result is immediate by definition, and the argument is trivial if φ is a conjunction or a negation. Thus, assume that φ is of the form $K_i\psi$. Let $\kappa = \sigma(P)$. Then $(\mathbf{I}(P, \gamma, \pi), r, m) \models K_i\psi$ iff $(\mathbf{I}(P, \gamma, \pi), r', m') \models \psi$ for all (r', m') such that $r' \in \mathbf{R}(P, \gamma)$ and $r'_i(m') = r_i(m)$. But $r' \in \mathbf{R}(P, \gamma)$ iff $\kappa(r') = 0$. Thus, $(\mathbf{I}(P, \gamma, \pi), r, m) \models K_i\psi$ iff $(\mathcal{I}^+(\gamma, \pi), r', m') \models \psi^B$ for all (r', m') such that $\kappa(r') = 0$ and $r'_i(m') = r_i(m)$. Note that $\mathsf{min}_i^\kappa(r, m) \ = 0$ (because $\kappa(r) = 0$). Thus, it easily follows that $(\mathbf{I}(P, \gamma, \pi), r, m) \models K_i\psi$ iff $(\mathcal{I}^+(\gamma, \pi), r, m) \models B_i\psi^B$. ∎

In light of Theorem 3.2, from this point on we work with the larger system $\mathcal{I}^+(\gamma, \pi)$ and use belief relative to deviation-compatible ranking generators, instead of working with the system $\mathbf{I}(P, \gamma, \pi)$ and using knowledge.

By having both ranking generators and order generators in our framework, we can handle both belief and counterfactual reasoning. Thus, for example, we can write $B_3(do(1, \mathsf{a}) > \varphi)$ to represent agent 3's belief that if agent 1 were to perform action a in the next round, then φ would hold. We can further write $B_3(do(1, \mathsf{a}) > \varphi) \quad > \quad \psi$ to state that were it the case that agent 3 had the above belief, then in fact ψ would hold. Arbitrary nesting of belief and counterfactuals is allowed. To take advantage of the expressive features of the framework, we now define the analogue of knowledge-base programs, to allow for belief and counterfactuals.

A *counterfactual belief-based program* (or *cbb program* for short) has the same form as a knowledge-based program, except that the underlying logical language for the formulas appearing in tests is now $\mathcal{L}_B^>$ instead of $\mathcal{L}_K$, and all tests in the program text Pg_i for agent i are formulas of the form $B_i\psi$ or $\neg B_i\psi$. As with knowledge-based programs, we are interested in when a protocol P *implements* a cbb program Pg_{cb}. Again, the idea is that the protocol should act according to the high-level program, when the tests are evaluated relative to the counterfactual belief-based system corresponding to P. To make this precise, given an extended system $\mathcal{J} = (\mathcal{I}, \lessdot, \kappa)$ and a cbb program Pg_{cb}, let $\mathsf{Pg}_{cb}^{\mathcal{J}}$ denote the protocol derived from Pg_{cb} by using $\mathcal{J}$ to evaluate the belief tests. That is, a test in Pg_{cb} such as $B_i\varphi$ holds at a point (r, m) relative to $\mathcal{J}$ if φ holds at all points (r', m') in $(\mathcal{I}, \kappa)$ such that $r'_i(m') = r_i(m)$ and $\kappa(r') = \mathsf{min}_i^{\kappa}(r, m)$. Define an *extended context* to be a tuple (γ, π, o, σ), where (γ, π) is an interpreted context, o is an ordering generator for $\mathcal{R}^+(\gamma)$, and σ is a deviation-compatible ranking generator for γ. An extended system $(\mathcal{I}, \lessdot, \kappa)$ *represents* the belief-based program Pg_{cb} in extended context (γ, π, o, σ) if (a) $\mathcal{I} = \mathcal{I}^+(\gamma, \pi)$, (b) $\lessdot = o(\mathsf{Pg}_{cb}^{(\mathcal{I}, \lessdot, \kappa)})$, and (c) $\kappa = \sigma(\mathsf{Pg}_{cb}^{(\mathcal{I}, \lessdot, \kappa)})$. A protocol P *implements* Pg_{cb} in (γ, π, o, σ) if $P = \mathsf{Pg}_{cb}^{(\mathcal{I}^+(\gamma,\pi), o(P), \sigma(P))}$. Protocol P *de facto implements* Pg_{cb} in (γ, π, σ) if $P \approx_\gamma \mathsf{Pg}_{cb}^{(\mathcal{I}^+(\gamma,\pi), o(P), \sigma(P))}$.

There is a close connection between the notions of implementation for knowledge-based programs and implementation for cbb programs using deviation-compatible rankings. Given a knowledge-based program Pg_{kb}, we denote by Pg_{kb}^{B} the program that results from replacing every knowledge operator K_i appearing in Pg_{kb} to B_i, for all agents $i = 1, \ldots, n$. (This is, in particular, a cbb programs with no counterfactual operators.)

Theorem 3.3 *Let Pg_{kb} be a knowledge-based program and let σ be a deviation-compatible ranking generator for γ. Moreover, let o be an arbitrary ordering generator for $\mathcal{R}^+(\gamma)$. A protocol P de facto implements Pg_{kb} in (γ, π) if and only if P de facto implements Pg_{kb}^{B} in (γ, π, o, σ).*

Proof Since σ is deviation compatible, by Theorem 3.2, for all points (r, m) of $\mathbf{R}(P, \gamma)$, we have that $(\mathbf{I}(P, \gamma, \pi), r, m) \models \varphi$ iff $(\mathcal{I}^+(\gamma, \pi), o(P), \sigma(P), r, m) \models \varphi^B$. Let $\mathsf{Pg}_{cb} = \mathsf{Pg}_{kb}^{B}$ and let $\mathcal{J}(P) = (\mathcal{I}^+(\gamma, \pi), o(P), \sigma(P))$. Then

$$(\mathsf{Pg}_{kb})_i^{\mathbf{I}(P,\gamma,\pi)}(r_i(m)) = (\mathsf{Pg}_{cb})_i^{\mathcal{J}(P)}(r_i(m)) \text{ whenever } r \in \mathbf{R}(P, \gamma). \tag{1}$$

Now suppose that P de facto implements Pg_{kb}. By definition, $P \approx_\gamma \mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$. Thus, the only global states that arise when running $\mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}$ are those of the form $r(m)$ for some $r \in \mathbf{R}(P, \gamma)$. It easily follows from (1) that $\mathcal{I}(\mathsf{Pg}_{kb}^{\mathbf{I}(P,\gamma,\pi)}, \gamma, \pi) = \mathcal{I}(\mathsf{Pg}_{cb}^{\mathcal{J}(P)}, \gamma, \pi)$. Thus, P de facto implements Pg_{cb} as well. The argument in the other direction is analogous. ▮

Theorem 3.3 shows that a protocol P de facto implements a knowledge-based program iff P de facto implements the corresponding belief-based program. Thus, by using deviation-compatible rankings, cbb programs can essentially emulate knowledge-based programs. The move to cbb programs as defined here thus provides what may be considered a conservative extension of the knowledge-based framework: it allows us to treat beliefs *and* counterfactuals, while being able to handle everything that the old theory gave us without changing the results.

3.3 Analysis of the Bit-Transmission Problem

Recall the program BT''_S from the introduction: **if** $K_S(\mathit{recbit})$ **then** skip **else** $\mathsf{sendbit}$. With this program, S keeps sending the bit until it knows that R has received the bit. As discussed in the introduction, it would be even more efficient for S to stop sending the bit once it knows that *eventually* R will receive it. As we saw, replacing $K_S(\mathit{recbit})$ by $K_S(\Diamond \mathit{recbit})$ leads to problems. We can deal with these problems by using counterfactuals (and, thus, belief rather than knowledge), as in the cbb program $\mathsf{BT}^{>}_S$ from the introduction:

$$\textbf{if } B_S(do(S,\mathsf{skip}) > \Diamond \mathit{recbit}) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

This program says that S should send the bit unless it believes that *even if it would not send the bit in the current round*, R would eventually receive the bit. Similarly, the program $\mathsf{BT}^{\Diamond B}_S$ says that S should send the bit unless it believes that R would eventually correctly believe its value:

$$\textbf{if } B_S(do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit})) \textbf{ then } \mathsf{skip} \textbf{ else } \mathsf{sendbit}.$$

(Recall that $B_R(\mathit{bit})$ is short for $(\mathit{bit} = 0 \wedge B_R(\mathit{bit} = 0)) \vee (\mathit{bit} = 1 \wedge B_R(\mathit{bit} = 1))$.)

Let $\mathsf{BT}^{>} = (\mathsf{BT}^{>}_S, \mathsf{SKIP}_R)$ and, similarly, let $\mathsf{BT}^{\Diamond B} = (\mathsf{BT}^{\Diamond B}_S, \mathsf{SKIP}_R)$. We now consider the implementations of $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$ in three different contexts:

- γ_1, in which messages are guaranteed to be delivered within five rounds;[9]
- γ_2, in which messages are guaranteed to arrive eventually, but there is no upper bound on message delivery time; and
- γ_3, in which a message that is sent infinitely often is guaranteed to arrive, but there is no upper bound on message delivery time. (Nothing can be said about a message sent only finitely often; this is a standard type of *fairness* assumed in the literature [Francez 1986].)

In all contexts that we consider, messages cannot be reordered or duplicated. Moreover, a message can be delivered only if it was previously sent. We assume for now that we are working in synchronous systems, so that processes can keep track of the round number. (Indeed, we cannot really make sense out of messages being delivered in five rounds in asynchronous systems.) At the end of this section we briefly comment on how our results can be modified to apply to asynchronous systems. We now describe these contexts more formally.

In $\gamma_1 = (P^1_e, \mathcal{G}^1_0, \tau^1, \Psi^1)$, an agent can perform one of two actions: skip and $\mathsf{sendbit}$, with the obvious outcome. The local state of S consists of three components: (a) a

[9] There is nothing special about five rounds here; another other fixed number would do for the purposes of this example.

Boolean variable *bit* that is fixed throughout the run, (b) a clock value, encoded in the variable *time*, which is always equal to the round number; at a point (r, m) the clock value is m, and (c) the *message history*, which is the sequence of messages that S has sent and received, each marked by time at which it was sent or received. The local state of the receiver R consists of the clock value and R's message history. Assume that the set $\mathcal{G}_0^1$ of initial states in γ_1 consists of two states—one in which $bit = 0$ and one in which $bit = 1$. In both states the clock values are 0 and message histories are empty. In this context, messages are guaranteed to be delivered within at most five rounds. The environment can perform the action of delivering a message. Its protocol P_e^1 consists of deciding when messages are delivered, subject to this constraint. Since the environment's state keeps track of all actions performed, it can be determined from the state which messages are in transit and how long they have been in transit. Ψ^1 makes no restrictions: all runs are considered admissible.

The context $\gamma_2 = (P_e^2, \mathcal{G}_0^2, \tau^2, \Psi^2)$ is a variant of γ_1 with asynchronous communication. $\mathcal{G}_0^2 = \mathcal{G}_0^1$, and the local states of S and R are the same as in γ_1. Every message sent is guaranteed to be delivered, but there is no bound on the time it will spend in transit. Thus, the environment's state again keeps track of the messages in transit, while the environment's protocol P_e^2 decides at each point (nondeterministically) which, if any, of the messages in transit should be delivered in the current round. The constraint that messages are guaranteed to eventually be delivered is captured by the admissibility constraint Ψ^2; the set Ψ^2 consists of the runs in which every message sent is eventually delivered.

The only difference between $\gamma_3 = (P_e^3, \mathcal{G}_0^3, \tau^3, \Psi^3)$ and γ_2 is that the admissibility condition Ψ^3 is more liberal than (i.e., is a superset of) Ψ^2. The set Ψ^3 consists of all runs r that are fair in the sense that, for every time m, if a given message μ is sent infinitely often in r after time m, then at least one of the copies of μ sent after time m is delivered.

We define three sets of extended contexts, extending γ_i, $i = 1, 2, 3$. Let EC_i consist of all contexts of the form $(\gamma_i, \pi, o, \sigma)$, $i = 1, 2, 3$, where π interprets the propositions $bit = 0$ and $bit = 1$ in the natural way, o respects protocols, and σ is deviation compatible.

We claim that both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\diamond B}$ solve the bit-transmission problem in every extended context in EC_i, $i = 1, 2, 3$. But what does it mean for a protocol to "solve" the bit-transmission problem? To make this precise, we need to specify the problem. In the case of the bit-transmission problem, the specification is simple: we want the receiver to eventually know the bit. Thus, we say that a cbb-program Pg solves the bit-transmission problem in extended context $\zeta = (\gamma, \pi, o, \sigma)$ if, for every protocol P that de facto implements Pg, we have that $(\mathcal{J}^+(P, \zeta), r, 0) \models \diamond B_R(bit)$ for every run $r \in \mathbf{R}(P, \gamma)$. Notice that using belief here is safe, because we are requiring only that the belief hold in runs of P. Lemma 3.1 guarantees that, in these runs (which all have rank 0), the beliefs are true.

Theorem 3.4 *Both* $\mathsf{BT}^{>}$ *and* $\mathsf{BT}^{\diamond B}$ *solve the bit-transmission problem in all the ex-*

tended contexts $EC_1 \cup EC_2 \cup EC_3$.

Proof Let $\zeta = (\gamma, \pi, o, \sigma)$ be a context in $EC_1 \cup EC_2 \cup EC_3$ and assume that P de facto implements $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$ in ζ. Let $\mathcal{J} = (\mathcal{I}(P, \gamma), o(P), \sigma(P))$ and let $r \in \mathbf{R}(P, \gamma)$ be a run of P in γ. We first consider the case that P implements $\mathsf{BT}^{>}$; the argument in the case that P implements $\mathsf{BT}^{\Diamond B}$ is even easier, and is sketched afterwards. There are two cases:

(a) Suppose that $(\mathcal{J}, r, m) \models B_S(do(S, \mathsf{skip}) > \Diamond recbit)$ for some $m > 0$. Since P de facto implements $\mathsf{BT}^{>}$, S performs skip in round $m+1$ of r. Thus, we have that $(\mathcal{J}, r, m) \models do(S, \mathsf{skip})$. Since $\sigma(P)$ is deviation compatible and $r \in \mathbf{R}(P, \gamma)$, it follows that $(\mathcal{J}, r, m) \models do(S, \mathsf{skip}) > \Diamond recbit$. Since o respects protocols, $(r, m) \in \mathtt{closest}([\![do(S, \mathsf{skip})]\!], (r, m), \mathcal{J})$. It now follows from the semantics of $>$ that $(\mathcal{J}, r, m) \models \Diamond recbit$. Since P de facto implements $\mathsf{BT}^{>}$, if S sends a value in a run r' of P, S is actually sending the bit. Since $\sigma(P)$ is deviation compatible, it follows that in every run r' of P, we have that $(\mathcal{J}, r', m') \models recbit \Rightarrow B_R(bit)$, since all the points in $\mathsf{min}_R(r', m')$ are points on runs of P. Thus, $(\mathcal{J}, r, m) \models B_R(bit)$.

(b) Suppose that $(\mathcal{J}, r, m) \not\models B_S(do(S, \mathsf{skip}) > \Diamond recbit)$ for all $m \geq 0$. Since P de facto implements $\mathsf{BT}^{>}$, it follows that S sends the bit in every round of r. (In particular, the bit is sent by S infinitely often.) All three contexts under consideration have the property that a message sent infinitely often is guaranteed to be delivered. Thus, at some time $m' \geq 0$ in r, the receiver will receive the bit; that is, $(\mathcal{J}, r, m') \models recbit$ for some $m' > 0$. we have by Then, just as in part (a), it follows that $(\mathcal{J}, r, m') \models B_R(bit)$, and hence that $(\mathcal{J}, r, 0) \models \Diamond B_R(bit)$.

The argument is almost identical (and somewhat simpler) if P implements $\mathsf{BT}^{\Diamond B}$. Now we split into two cases according to whether there is some m such that $(\mathcal{J}, r, m) \models B_S(do(S, \mathsf{skip}) > \Diamond B_R(bit))$. Using the same arguments as above (but skipping the argument that $\mathcal{J} \models recbit \Rightarrow B_R(bit)$) we get that, in both cases, $(\mathcal{J}, r, 0) \models \Diamond B_R(bit)$. ∎

Theorem 3.4, while useful, does not give us all we want. In particular, it shows neither that $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$ is implementable nor that S sends relatively few messages according to any protocol that implements $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$ (which, after all, was the goal of using counterfactuals in this setting). In fact, as we now show, both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$ are implementable in all three sets of contexts, and their implementations are as message-efficient as possible. We consider each of EC_1, EC_2,and EC_3 in turn.

Intuitively, in order to solve the bit-transmission problem in a context in which messages are always delivered, sending the bit only once in any given run should suffice. Consider the collection of protocols $P^1(k, m) = (P^1_S(k, m), \mathsf{SKIP}_R)$ for $k, m \in \mathbf{N}$, where $P^1_S(k, m)$ is described by the program

$$\textbf{if } (time = k \text{ and } bit = 0) \text{ or } (time = m \text{ and } bit = 1) \textbf{ then } \mathsf{sendbit} \textbf{ else } \mathsf{skip}.$$

In these protocols, the sender S sends its bit at time k if the bit value is 0, and at time m if it is 1. We now show that all protocols of the form $P^1(k, m)$ implement $\mathsf{BT}^>$ in all contexts in EC_1:

Lemma 3.5 *The protocol $P^1(k, m)$ de facto implements $\mathsf{BT}^>$ in every extended context in EC_1.*

Proof Fix k, m, and a context $\zeta = (\gamma_1, \pi, o, \kappa) \in EC_1$. We want to show that $P^1(k, m) \approx_{\gamma_1} (\mathsf{BT}^>)^{\mathcal{J}(k,m)}$, where $\mathcal{J}(k, m) = (\mathcal{I}^+(\gamma_1, \pi_1), o(P^1(k, m)), \sigma(P^1(k, m)))$. We can characterize a run consistent with $P^1(k, m)$ by the value of *bit* and when the one message sent by S is received. Let $r_{b,n}$ be the run where $bit = b$ and the message is received at time n (clearly $k + 5 \geq n > k$ if $b = 0$ and $m + 5 \geq n > m$ if $b = 1$). Clearly the formula *recbit* holds in run $r_{b,n}$ from time n on. Thus, $\Diamond recbit$ holds at every point in every run consistent with $P^1(k, m)$ in the system $\mathcal{J}(k, m)$. Note that the runs $r_{b,n}$ are precisely those of rank 0 in $\mathcal{J}(k, m)$.

We now show that a run r is consistent with $(\mathsf{BT}^>)^{\mathcal{J}(k,m)}$ in γ_1 iff $r = r_{b,n}$ for $b \in \{0, 1\}$ and a value of n satisfying $k + 5 \geq n > k$ if $b = 0$ and $m + 5 \geq n > m$ if $b = 1$. So suppose that r is consistent with $(\mathsf{BT}^>)^{\mathcal{J}(k,m)}$ and the value of the bit in r is 0. It suffices to show that S sends exactly one message in r, and that happens at time k. If $n' \neq k$, then clearly $(\mathcal{J}(k, m), r, n') \models (S, \mathsf{skip}) > \Diamond recbit$, since the closest point to (r, n') where $do(S, \mathsf{skip})$ holds is (r, n') itself. On the other hand, if $n' = k$, then $\mathtt{closest}([\![do(S, \mathsf{skip})]\!], (r, n'), \mathcal{J}(k, m)) = \{(r'_0, n')\}$, where r'_0 is the run where S never sends any messages and the initial bit is 0. In this case, the properties of γ_1 guarantee that no message is ever received by R in r', and $\Diamond recbit$ does not hold at (r', k). It follows that the test $B_S(do(S, \mathsf{skip}) > \Diamond recbit)$ fails at (r, k), and r is consistent with $\mathsf{BT}^>$ if and only if the action *sendbit* is performed in round $k + 1$ of r. Hence, r is one of the runs $r_{0,n}$ with $k + 5 \geq n > k$. A completely analogous treatment applies if $bit = 1$ in r. We thus have that exactly the runs $r_{b,n}$ described are consistent with $(\mathsf{BT}^>)^{\mathcal{J}(k,m)}$ in γ_1, and hence $P^1(k, m)$ de facto implements $\mathsf{BT}^>$ in every extended context in EC_1, as desired. ∎

In the context γ_1, there is a fixed bound on message delivery time. As a result, we might hope to save on message delivery in some cases. Suppose that we use a one-sided protocol, that sends the bit only if $bit = 0$. Then the receiver should be able to conclude that the value of the bit is 1 if a message stating the bit is 0 does not arrive within the specified time bounds. More generally, define the collection of protocols $P^2(k, b) = (P^2_S(k, b), \mathsf{SKIP}_R)$ for $b \in \{0, 1\}$ and $k \in \mathbf{N}$, where $P^2_S(k, b)$ is the protocol implementing the program

$$\textbf{if } time = k \text{ and } bit = b \textbf{ then } \mathsf{sendbit} \textbf{ else } \mathsf{skip}.$$

According to $P^2_S(k, b)$, the sender S sends a message only in runs where the bit is b; if the bit is $1 - b$, it sends no messages. Moreover, in runs where the bit is b, S sends only one message, at time k. This type of optimization (sending a message only for one of the two bit values) was used in the message-optimal protocols of [Hadzilacos and Halpern 1993]; it can be used in synchronous systems in which there is an upper bound on the message delivery time, as in contexts in EC_1.

It is easy to verify that $P^2(k,b)$ does not implement $\mathsf{BT}^{>}$: Intuitively, in a run r of $P^2(k,b)$ with $bit = 1-b$, the sender S never sends the bit, and hence $\Diamond recbit$ does not hold. Since S follows $P^2(k,b)$ in r, the formula $do(S,\mathsf{skip})$ holds at time 0 in r. It follows that in evaluating the test $B_S(do(S,\mathsf{skip}) > \Diamond recbit)$ the closest point to $(r,0)$ is $(r,0)$ itself. Because $\Diamond recbit$ does not hold at that point, the test fails, and according to $\mathsf{BT}^{>}$ the sender S should perform $\mathsf{sendbit}$. Since, in fact, S does not perform $\mathsf{sendbit}$ at $(r,0)$, and r is a run of $P^2(k,b)$, we conclude that $P^2(k,b)$ does not implement $\mathsf{BT}^{>}$. However, as we now show, $P^2(k,b)$ does implement the more sophisticated program $\mathsf{BT}^{\Diamond B}$:

Lemma 3.6 *Every instance of $P^2(k,b)$ de facto implements $\mathsf{BT}^{\Diamond B}$ in every context in EC_1.*

Proof Fix k, b, and a context $\zeta = (\gamma_1,\pi,o,\sigma) \in EC_1$. We want to show that $P^2(k,b) \approx_{\gamma_1} (\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,b)}$, where $\mathcal{J}(k,b) = (\mathcal{I}^+(\gamma_1,\pi), o(P^2(k,b)), \sigma(P^2(k,b)))$. Note that there are exactly six runs consistent with $P^2(k,b)$ in context γ_1: five runs r_b^m, $m = k+1,\ldots,k+5$, where the value of the bit is b, the message is sent in round $k+1$ and it arrives in round m; the sixth run is r_{1-b}, where the value of the bit is $1-b$ and no message is sent. It is easy to check that in the extended system $\mathcal{J}(k,b)$, the formula $bit = b \wedge B_R(bit = b)$ holds in runs r_b^m from time m on, while in run r_{1-b} the formula $bit = 1-b \wedge B_R(bit = 1-b)$ holds from time $k+5$ on. Thus, $\Diamond B_R(bit)$ holds at every point in the six runs in $\mathbf{R}(P^2(k,b),\gamma_1)$. Note that these six runs are exactly the runs of rank 0.

We now show that r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,b)}$ iff $r \in \mathbf{R}(P^2(k,b),\gamma_1)$. We consider two cases, according to the values of the bit in r. First suppose that $bit = 1-b$ in the run r. We prove by induction on $m' \geq 0$ that (a) if r is consistent with $\mathsf{BT}_S^{\mathcal{J}(k,b)}$ then (i) $r(m') = r_{1-b}(m')$ and (ii) $(\mathsf{BT}^{\Diamond B})_S^{\mathcal{J}(k,b)}(r_S(m')) = \mathsf{skip}$, and (b) r_{1-b} is consistent with $(\mathsf{BT}^{\Diamond B})_S^{\mathcal{J}(k,b)}$ up to time m'. For the base case, observe that $r(0) = r_{1-b}(0)$ because there is only one initial state in γ_1 with $bit = 1-b$. Clearly r_{1-b} is consistent with $(\mathsf{BT}^{\Diamond B})_S^{\mathcal{J}(k,b)}$ up to time 0. Thus, parts (a)(i) and (b) hold. For part (a)(ii), to see that $(\mathsf{BT}^{\Diamond B})_S^{\mathcal{J}(k,b)}(r_S(0)) = \mathsf{skip}$, it suffices to show that $(\mathcal{J}(k,b),r,0) \models B_S(do(S,\mathsf{skip}) > \Diamond B_R(bit))$. Since σ_1 is deviation compatible and S knows that $bit = 1-b$, it follows that $\mathsf{min}_S^{\sigma(P^2(k,b))}(r,0) = \{(r_{1-b},0)\}$. Thus, it suffices to show that $(\mathcal{J}(k,b),r_{1-b},0) \models do(S,\mathsf{skip}) > \Diamond B_R(bit)$. But this is immediate from the fact that $(\mathcal{J}(k,b),r_{1-b},0) \models do(S,\mathsf{skip})$ and, as observed earlier, that $(\mathcal{J}(k,b),r_{1-b},0) \models \Diamond B_R(bit)$.

For the inductive step in the case $bit = 1-b$, assume that the inductive claim holds for time $m' \geq 0$. We want to show that it holds at time $m'+1$. Part (a)(i) and (b) are immediate from the inductive hypothesis. The argument for part (a)(ii) is the same as in the base case. This completes the inductive argument. It follows immediately from the induction that r_{1-b} is consistent with $\mathsf{BT}_S^{\mathcal{J}(k,b)}$ and that if r is consistent with $\mathsf{BT}_S^{\mathcal{J}(k,b)}$ and $\mathit{bit} = 1-b$ in r, then $r = r_{1-b}$.

Now consider the case where $\mathit{bit} = b$ in r. Define b-runs to be the set $\{r_{k+1}, r_{k+2} \ldots, r_{k+5}\}$, and b-pts(m') to be $\{(r_{k+1}, m'), (r_{k+2}, m'), \ldots, (r_{k+5}, m')\}$. We show by induction on $m' \geq 0$ that if r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,b)}$, then

(a) $r(m') \in \mathsf{b}$-pts(m'),

(b) $$(\mathsf{BT}_S^{\Diamond B})^{\mathcal{J}(k,b)}(r_S(m')) = \begin{cases} \mathsf{skip} & \text{if } m' \neq k \\ \mathsf{sendbit} & \text{if } m' = k, \end{cases}$$

(c) at least one run in b-runs agrees with r up to time m'; moreover, if $m' \geq k+5$, then exactly one run in b-runs agrees with r up to time m'.

For the base case, it is again immediate that $r(0) \in \mathsf{b}$-pts(0) and that all runs in b-runs agree with r up to time 0. To see that part (b) holds, first note that $\mathsf{min}_S^{\sigma(P^2(k,b))}(r,0) = \{(r^{k'}, 0) : k' = 1, \ldots, 5\}$. There are now two cases: if $k = 0$ (so that S sends a message in round 1 of all the runs in b-runs), then we must show that $(\mathcal{J}(k,b), r, 0) \models \neg B_S(\mathit{do}(S, \mathsf{skip}) > \Diamond B_R(\mathit{bit}))$, so that $\mathsf{BT}_S^{\mathcal{J}(k,b)}(r_S(0)) = \mathsf{sendbit}$. Note that, if $k = 0$, then $\mathtt{closest}([\![\mathit{do}(S,\mathsf{skip})]\!], (r^{k'}, 0), \mathcal{J}(k,b)) = \{r^*\}$ for $k' = 1, \ldots, 5$, where r^* is the run where $\mathit{bit} = b$ and no messages are ever sent by S or R. Thus, it suffices to show that $(\mathcal{J}(k,b), r*, 0) \models \neg\Diamond B_R(\mathit{bit})$. It is easy to see that, since σ_1 is deviation compatible, we must have $(r_{1-b}, m) \in \mathsf{min}_R^{\sigma(P^2(k,b))}(r^*, m)$, for all $m \geq 0$. Thus, $(\mathcal{J}, r^*, m) \not\models \mathit{bit} = 1-b \wedge B_R(\mathit{bit} = 1-b)$ for all $m \geq 0$, and hence $(\mathcal{J}, r^*, m') \models \neg\Diamond B_R(\mathit{bit})$ for all $m' \geq 0$, as desired. On the other hand, if $k > 0$, we must show that $(\mathcal{J}(k,b), r, 0) \models B_S(\mathit{do}(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit}))$. Note that if $k > 0$, then $\mathtt{closest}([\![\mathit{do}(S,\mathsf{skip})]\!], (r^{k'}, 0), \mathcal{J}(k,b)) = \{r^{k'}\}$, for $k' = 1, \ldots, 5$. Since $(\mathcal{J}(k,b), r^{k'}, 0) \models \mathit{do}(S,\mathsf{skip}) \wedge B_R(\mathit{bit}))$, we are done.

The argument in the inductive step is almost identical, except that it now breaks into the cases $m' < k$, $m' = k$, $k < m' < k+5$, and $m' \geq k+5$. We leave details to the reader.

Finally, we must show that each run $r \in \mathsf{b}$-runs is consistent with $(\mathsf{BT}_S^{\Diamond B})^{\mathcal{J}(k,b)}$. We proceed by induction on m' to show that r is consistent with $(\mathsf{BT}_S^{\Diamond B})^{\mathcal{J}(k,b)}$ up to time m'. This involves proving part (b) of the induction above for each $r \in \mathsf{b}$-runs. The proof is similar to that above, and left to the reader. ▮

The preceding discussion has shown that $P^2(k,b)$ implements $\mathsf{BT}^{\Diamond B}$, but not $\mathsf{BT}^{>}$, in contexts in EC_1. Lemma 3.5 shows that $P^1(k,m)$ implements $\mathsf{BT}^{>}$ in contexts in EC_1. An obvious question is whether $P^1(k,m)$ implements $\mathsf{BT}^{\Diamond B}$ in contexts in EC_1. We now show that if $k \neq m$, then $P^1(k,m)$ does *not* implement $\mathsf{BT}^{\Diamond B}$; if $k = m$, then whether $P^1(k,m)$ implements $\mathsf{BT}^{\Diamond B}$ depends on what the receiver believes in runs where he does not receive a message. Since there is no run of $P^1(k,m)$ where the receiver receives no messages, this is not determined by just assuming that we have a deviation-compatible ranking generator. Given a ranking κ, let $\kappa(n,b)$ be the rank of the run with least rank where (a) the receiver does not receive any messages up to and including time n and (b) the bit has value b. We say that a ranking κ is *biased* if $\kappa(n,0) \neq \kappa(n,1)$ holds for at least one time instant n. Note that if $\kappa(n,i) < \kappa(n,i \oplus 1)$ then, in the absence of messages, R will believe that the bit is i at time n.

Lemma 3.7 *Let $\zeta = (\gamma_1, \pi, o, \sigma) \in EC_1$. The protocol $P^1(k,m)$ de facto implements $\mathsf{BT}^{\Diamond B}$ in ζ exactly if both (a) $k = m$ and (b) $\sigma(P^1(k,k))$ is not biased.*

Proof Fix a context $\zeta = (\gamma_1, \pi, o, \sigma) \in EC_1$. As in the proof of Lemma 3.5, define $\mathcal{J}(k,m) = (\mathcal{I}^+(\gamma_1,\pi), o(P^1(k,m)), \sigma(P^1(k,m)))$ and the runs $r_{b,n}$.

First suppose that $\sigma(P^1(k,k))$ is not biased. We show that $P^1(k,k)$ de facto implements $\mathsf{BT}^{\Diamond B}$ in ζ. By definition, in each of the ten runs $r_{b,n}$ of rank 0 in the extended system $\mathcal{J}(k,k)$, *recbit* holds at the time n when the receiver R receives the bit. Since R receives the correct bit, it is easy to see that in fact $(\mathcal{J}(k,k), r_{b,n}, n) \models B_R(\mathit{bit})$. Thus, $\Diamond B_R(\mathit{bit})$ holds at every point in the ten runs of the form $r_{b,n}$ in the system $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$. Moreover, $(\mathcal{J}(k,k), r_{b,n}, m) \models do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit})$ for $m \neq k$. Since the runs $r_{b,n}$ are the runs of rank 0, it actually follows that $(\mathcal{J}(k,k), r_{b,n}, m) \models B_S(do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit}))$ for $m \neq k$. We now show that $(\mathcal{J}(k,k), r_{b,n}, k) \models \neg B_S(do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit}))$. Note that $\mathtt{closest}([\![do(S,\mathsf{skip})]\!], (r_{b,n},k), \mathcal{J}(k,k)) = \{(r'_b,k)\}$, where r'_b is the run where the bit is b and S sends no messages. Suppose that $(\mathcal{J}(k,k), r'_b, k) \models \Diamond B_R(\mathit{bit} = b)$. Thus, there is some $n \geq k$ such that $(\mathcal{J}(k,k), r'_b, n) \models B_R(\mathit{bit} = b)$. Then we must have $\kappa(n,b) < \kappa(n, b \oplus 1)$, so that κ is biased, contradicting the assumption. Thus, $(\mathcal{J}(k,k), r'_b, k) \models \neg\Diamond B_R(\mathit{bit} = 0)$, so $(\mathcal{J}(k,k), r_{b,n}, k) \models \neg B_S(do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit}))$, as desired. In this case, by $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$, the sender S should perform $\mathsf{sendbit}$ at time k. It follows that $r_{b,n}$ is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$.

We next show that if r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$, then $r \in \{r_{b,n} : b = 0,1,\ n = k+1, \ldots, k+5\}$. So suppose that the bit is 0 in r and that r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$. Just as in the proof of Lemma 3.6, it is easy to show by induction on m that no messages are sent in r at time $m < k$: It is easy to see that $(\mathcal{J}(k,k), r, m) \models B_S(do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit}))$ for $k < m$, since $(r,m) \sim_R (r_{b,n}, m)$. Just as in the case of $r_{b,n}$, we can show that $(\mathcal{J}(k,k), r, k) \models \neg B_S(do(S,\mathsf{skip}) > \Diamond B_R(\mathit{bit}))$. Thus, since r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$, the sender S sends a message at time k in r. It is easy to show that S does not send the bit after time k; we leave details to the reader. Thus, if r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$ then S sends the bit in r at time k (and does not send it at any other time), so r is of the form $r_{b,n}$.

We next claim that if $k \neq m$ then $P^1(k,m)$ does not de facto implement $\mathsf{BT}^{\Diamond B}$ in ζ. Without loss of generality, suppose that $k < m$. By the properties of γ_1, messages can take up to five time units to be delivered. Hence, there is a run of $P^1(k,m)$ with $\mathit{bit} = 1$ in which the sender's message is not delivered by time $m + 4$. However, because $k < m$, there is no run with $\mathit{bit} = 0$ where no message is delivered by time $m + 4$. Because σ is deviation compatible, it follows that $\kappa(m+4,1) = 0 < \kappa(m+4,0)$. Thus, $(\mathcal{J}(k,m), r_{1,m+4}, m+4) \models B_R(\mathit{bit} = 1)$, so $(\mathcal{J}(k,m), r_{1,m+j}, m) \models B_S(do(S,\mathsf{skip}) >$

$\Diamond B_R(bit))$ for $j = 1, \ldots, 5$. Therefore, S should not send the bit at time m according to $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,m)}$ in runs where the bit is 1, showing that $P^1(k,m)$ does not de facto implement $\mathsf{BT}^{\Diamond B}$.

To complete the proof of the lemma, we need to show that if $\kappa = \sigma(P^1(k,k))$ is biased, then $P^1(k,k)$ does not implement $\mathsf{BT}^{\Diamond B}$ in ζ. So suppose that $\kappa = \sigma(P^1(k,k))$ is biased. Since κ is biased, there is an n for which $\kappa(n,0) \neq \kappa(n,1)$. Without loss of generality, assume that $\kappa(n,0) < \kappa(n,1)$. We must have $n > k$, since $\kappa(\ell,0) = \kappa(\ell,1) = 0$ for all $\ell \leq k$, because in all runs consistent with $P^1(k,k)$, the receiver R receives no messages up to time ℓ. It follows that $(\mathcal{J}(k,k), r, k) \models \Diamond B_R(bit = 0)$ for all runs r consistent with $P^1(k,k)$. Thus, $(\mathcal{J}(k,k), r_{0,k+j}, m) \models B_S(do(S, \mathsf{skip}) > \Diamond B_R(bit))$ for $j = 1, \ldots, 5$. It follows that, in runs where the bit is 0, S should not send the bit according to $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}(k,k)}$. This again establishes that $P^1(k,k)$ does not de facto implement $\mathsf{BT}^{\Diamond B}$. ∎

Now consider the context γ_2. Here there is no upper bound on message delivery times. As a result, S must send R messages regardless of what bit value is.

Lemma 3.8 *Every instance of $P^1(k,m)$ de facto implements both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$ in every context in EC_2.*

Proof The proof for the case of $\mathsf{BT}^{>}$ is identical to the proof given for contexts in EC_1 in Lemma 3.5. There are now infinitely many runs $r_{b,n}$ consistent with $P^1(k,m)$ rather than ten runs, but the argument remains sound. We leave details to the reader.

In the case of $\mathsf{BT}^{\Diamond B}$, the argument follows the same lines as the proof Lemma 3.5, except that the role of $\Diamond recbit$ is now played by $\Diamond B_R(bit)$. Fix k, m, and a context $\zeta = (\gamma_2, \pi, o, \sigma) \in EC_2$. We want to show that $P^1(k,m) \approx_{\gamma_2} (\mathsf{BT}^{\Diamond B})^{\mathcal{J}'(k,m)}$, where $\mathcal{J}'(k,m) = (\mathcal{I}^+(\gamma_2,\pi), o(P^1(k,m)), \sigma(P^1(k,m)))$. It is easy to check that in the extended system $\mathcal{J}'(k,m)$, the formula $B_R(bit = b)$ holds in run $r_{b,n}$ from time n on. Thus, $\Diamond B_R(bit)$ holds at every point in every run consistent with $P^1(k,m)$ in the system $\mathcal{J}'(k,m)$. Note that the runs $r_{b,n}$ are precisely those of rank 0 in $\mathcal{J}'(k,m)$. Finally, note that if (r', n) is an arbitrary point in $\mathcal{J}'(k,m)$ with $n > \max(k,m)$ and no messages are sent in r' up to time n, then $(\mathcal{J}'(k,m), r', n) \models \neg B_R(bit = 0) \wedge \neg B_R(bit = 1)$, since there are runs consistent with $P^1(k,m)$ where no messages arrive up to time n and the bit can be either 0 or 1; for example, $(r_{0,n+1}, n) \sim_R (r', n)$ and $(r_{1,n+1}, n) \sim_R (r', n)$.

We now show that a run r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}'(k,m)}$ in γ_2 iff $r = r_{b,n}$ for $b \in \{0,1\}$ and $n \geq 0$. So suppose that r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}'(k,m)}$ and the value of the bit in r is 0. It suffices to show that S sends exactly one message in r, and that happens at time k. The argument is very similar to that in Lemma 3.6. If $n < k$, then clearly $(\mathcal{J}'(k,m), r, n) \models (S, \mathsf{skip}) > \Diamond B_R(bit)$, since the closest point to (r,n) where $do(S, \mathsf{skip})$ holds is (r,n) itself. On the other hand, if $n = k$, then $\mathtt{closest}([\![do(S,\mathsf{skip})]\!], (r,n), \mathcal{J}'(k,m)) = \{(r'_0, n)\}$, where r'_0 is the run where S sends no messages and the initial bit is 0. As observed earlier, we have $(\mathcal{J}'(k,m), r'_b, n) \models$

$\Box(\neg B_R(bit = 0) \wedge \neg B_R(bit = 1))$, so $(\mathcal{J}'(k,m), r_{b,n}, n) \models \neg(do(S, \mathsf{skip}) > \tilde{B}_R(bit))$. Thus, since r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}'(k,m)}$ in γ_2, S sends its bit at time k in r. Finally, if $n > k$, again we have $\mathtt{closest}([\![do(S,\mathsf{skip})]\!], (r,n), \mathcal{J}'(k,m)) = \{(r,n)\}$ so, again, S does not send a message at time n in r. Thus, r has the form $r_{0,n'}$ for some n'. The same argument shows that all runs of the form $r_{0,n'}$ are in fact consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}'(k,m)}$. The argument if $b = 1$ is identical (with m replacing k throughout). ∎

Finally, we consider the contexts in EC_3. In this case, communication is such that if R sends no messages, then S is guaranteed to have one of its messages delivered only in case it sends infinitely many message. This says that if we consider only protocols of the form (P_S, SKIP_R), then S must send infinitely many messages in every run. However, if a protocol sends infinitely many messages, then no particular one is necessary; if S does not send, say, the first message, then it still sends infinitely many, and R is guaranteed to get a message. This suggests that we will have difficulty finding a protocol that implements $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$. The following proposition prevides further evidence of this. If $I \subseteq I\!N$ (the natural numbers), let $P(I) = (P_S(I), \mathsf{SKIP}_R)$, where $P_S(I)$ is described by the program

$$\textbf{if } \mathit{time} \in I \textbf{ then } \mathsf{sendbit} \textbf{ else } \mathsf{skip}.$$

Thus, with $P_S(I)$, the sender S sends the bit at every time that appears in I.

Proposition 3.9 *No protocol of the form $P(I)$ de facto implements either $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$ in any context in EC_3.*

Proof We sketch the argument here and leave details to the reader. First suppose that I is finite. Let r be a run in $P(I)$ where none of the finitely many messages sent by S is received. Let $n = \sup(I) + 1$. Suppose that $(\gamma_3, \pi, o, \sigma) \in EC_3$. Let $\mathcal{J}(I) = (\mathcal{I}^+(\gamma_3, \pi), o(P(I)), \sigma(P(I)))$. Clearly, $\mathtt{closest}([\![do(S,\mathsf{skip})]\!], (r,n), \mathcal{J}(I)) = \{(r)\}$, since S performs the act skip at (r,n). However, since R never receives the bit in run r, and $\sigma(P(I))(r) = 0$, it follows that $(\mathcal{J}(I), r, n) \models \neg\Diamond \mathit{recbit}$ and $(\mathcal{J}(I), r, n) \models \neg B_R(bit)$. Thus, according to both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$, S should send a message at (r,n). It follows that $P(I)$ does not implement $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$.

Now suppose that I is infinite. The properties of γ_3 ensure that R does in fact receive the bit in every run of $P(I)$. Moreover, it is easy to check that when the message is received, both recbit and $B_R(bit)$ hold. Hence, for any given clock time $m \in I$, the formulas $do(S,\mathsf{skip}) > \Diamond \mathit{recbit}$ and $do(S,\mathsf{skip}) > \Diamond B_R(bit)$ hold at time m in all runs of the protocol. A straightforward argument shows that $\mathsf{sendbit}$ is neither compatible with $\mathsf{BT}^{>}$ nor with $\mathsf{BT}^{\Diamond B}$ at time m. ∎

Intuitively, Proposition 3.9 is a form of the “procrastinator’s paradox”: Any action that must be performed only eventually (e.g., washing the dishes) can always safely be postponed for one more day. Of course, using this argument inductively results in the action never being performed.

Despite Proposition 3.9, we now show that $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\diamond B}$ are both implementable in all contexts in EC_3. Let $P^\omega = (P^\omega_S, \mathsf{SKIP}_R)$, where P^ω_S is the protocol determined by the following program:

if *time* $= 0$ or $\mathsf{sendbit}$ was performed in the previous round, **then** $\mathsf{sendbit}$ **else** skip.

Since S's local state contains both the current time and a record of the time at which it sent every previous message, it can perform the test in $P^\omega(S)$. It is not too hard to see that P^ω is de facto equivalent to $P(I\!\!N)$ in γ_3—under normal circumstances the bit is sent in *each and every* round. The two protocols differ only in their counterfactual behavior. As a result, while $P(I\!\!N)$ implements neither $\mathsf{BT}^{>}$ nor $\mathsf{BT}^{\diamond B}$, the protocol P^ω implements both.

Lemma 3.10 *P^ω de facto implements both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\diamond B}$ in every context in EC_3.*

Proof We provide the proof for $\mathsf{BT}^{\diamond B}$. The proof for $\mathsf{BT}^{>}$ is similar, and left to the reader.

Fix a context $\zeta_3 = (\gamma_3, \pi, o, \sigma) \in EC_3$. We want to show that $P^\omega \approx_{\gamma_3} (\mathsf{BT}^{\diamond B})^{\mathcal{J}^\omega}$, where $\mathcal{J}^\omega = (\mathcal{I}^+(\gamma_3, \pi), o(P^\omega), \sigma(P^\omega))$. Let $R^\omega = \mathbf{R}(P^\omega, \gamma_3)$. Note that, for every natural number k, there are runs $r_{b,k} \in R^\omega$ in which $bit = b$ and no message that is sent by S in the first k rounds is ever delivered to R. It follows that if R has received no message by time m in run r of R^ω, then $(\mathcal{J}^\omega, r, m) \models \neg B_R(bit)$.

We now prove by induction on k that a run r is consistent with $(\mathsf{BT}^{\diamond B})^{\mathcal{J}^\omega}$ in γ_3 for k rounds exactly if S has performed $\mathsf{sendbit}$ in each of the first k rounds of r. The base case for $k = 0$ is vacuously true. For the inductive step, assume that the claim is true for $k = \ell$. Suppose that r is consistent with $(\mathsf{BT}^{\diamond B})^{\mathcal{J}^\omega}$ for $\ell+1$ rounds. By the induction hypothesis, the sender S has performed $\mathsf{sendbit}$ in each of the first ℓ rounds. Since r is, by assumption, consistent with $(\mathsf{BT}^{\diamond B})^{\mathcal{J}^\omega}$ for $\ell + 1$ rounds, S performs $\mathsf{sendbit}$ in round $\ell + 1$ of r exactly if $(\mathcal{J}^\omega, r, \ell) \models \neg B_S(do(S, \mathsf{skip}) > \diamond B_R(bit))$. Let $bit = b$ in r. Moreover, $\sigma_3(P^\omega)(r) = 0$ since σ_3 is deviation compatible. Clearly $(r, \ell) \sim_S (r_{b,\ell})$, where $r_{b,\ell} \in R^\omega$ is the run constructed earlier where none of the message sent by S in the first ℓ rounds arrive, since in both r and $r_{b,\ell}$, the bit is the same and S sends a message in each of the first ℓ rounds. Moreover, $\sigma(P^\omega)(r_{b,\ell}) = 0$, since σ is deviation compatible and $r_{b,\ell} \in R^\omega$. Thus, to show that $(\mathcal{J}^\omega, r, \ell) \models \neg B_S(do(S, \mathsf{skip}) > \diamond B_R(bit))$, it suffices to show that $(\mathcal{J}^\omega, r_{b,\ell}, \ell) \models \neg(do(S, \mathsf{skip}) > \diamond B_R(bit))$. The points in $\mathtt{closest}([\![do(S, \mathsf{skip})]\!], (r, \ell), \mathcal{J}^\omega)$ have the form (r', ℓ) where r' agrees with $r_{b,\ell}$ up to and including time ℓ, S does nothing in round ℓ of r', and S follows P^ω in all rounds after ℓ in r'. The key point here is that, by following P^ω, S sends no messages in r' after round ℓ. Consequently, in all runs appearing in this set of closest points, S sends a finite number of message (exactly ℓ, in fact). By the admissibility condition Ψ^3 of γ_3, there is one run in this set, which we denote by $\hat{r}$, in which R receives no messages. Note that $(\hat{r}, n) \sim_R (r_{0,n}, n)$ and $(\hat{r}, n) \sim_R (r_{1,n}, n)$, since in all of $\hat{r}$, $r_{0,n}$ and $r_{1,n}$, the receiver R receives no messages up to time n. Since both $r_{0,n}$ and $r_{1,n}$ are in

R^ω, it follows that they both have rank 0. Thus, $(\mathcal{J}^\omega, \hat{r}, n) \models \neg B_r(bit)$. That is, $B_R(bit)$ never holds in $\hat{r}$. It follows that $(\mathcal{J}^\omega, r_{b,\ell}, \ell) \models \neg(do(S, \mathsf{skip}) > \Diamond B_R(bit))$, as needed. We can thus conclude that r is consistent with $(\mathsf{BT}^{\Diamond B})^{\mathcal{J}^\omega}$ in γ_3 for $\ell + 1$ rounds exactly if S performs sendbit in the first $\ell + 1$ rounds, and we are done. ▮

Lemma 3.10 shows one way of resolving the procrastinator paradox: If one decides that an action (e.g., washing the dishes) that is not performed now will *never* be performed, then performing it becomes critical. (We are ignoring the issue of how one can "decide" to use such protocol. In the context of distributed computing, we can just make this the protocol; people are likely not to believe that this is truly the protocol.) In any case, using such a protocol makes performing the action consistent with the procrastinator's protocol of doing no more than what is absolutely necessary.

We can summarize our analysis of implementability of $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$ by the following theorem:

Theorem 3.11 *Both* $\mathsf{BT}^{>}$ *and* $\mathsf{BT}^{\Diamond B}$ *are de facto implementable in every extended context in* $EC_1 \cup EC_2 \cup EC_3$ *Moreover, if* P *de facto implements* $\mathsf{BT}^{>}$ *or* $\mathsf{BT}^{\Diamond B}$ *in a context* $\zeta \in EC_1 \cup EC_2$*, then* S *sends at most one message in every run consistent with* P *in* ζ*.*

Proof The implementability claims follow from Lemmas 3.5, 3.6, and 3.10. We now prove that S sends no more than one message in every run of a protocol that de facto implements $\mathsf{BT}^{>}$ or $\mathsf{BT}^{\Diamond B}$ in a context in $EC_1 \cup EC_2$. Suppose that $P = (P_S, P_R)$ de facto implements BT_R in $\zeta = (\gamma, \pi, o, \sigma) \in EC_1 \cup EC_2$. Further suppose, by way of contradiction, that there is a run r consistent with P in γ in which the sender sends more than one message. Suppose that the second message is sent at time k, and the value of the bit in r is b. Let $\mathcal{J} = (\mathcal{I}^+(\gamma, \pi), o(P), \sigma(P))$. Since $\gamma \in \{\gamma_1, \gamma_2\}$, all messages are guaranteed to arrive eventually in the context γ. Thus, it is easy to see that $(\mathcal{J}, r, k) \models B_S(\Diamond B_R(bit = b))$. It follows that $(\mathcal{J}, r, k) \models do(S, \mathsf{skip}) > B_R(bit)$. Since P is de facto consistent with $\mathsf{BT}^{>}$, this means that S should not send a message at (r, k). This is a contradiction. ▮

All the contexts we have considered are synchronous; the sender and receiver know the time. As we observed earlier, there is no analogue of γ_1 in the asynchronous setting, since it does not make sense to say that messages arrive in 5 rounds. However, there are obvious analogues of γ_2 and γ_3. Moreover, if we assume that S's local state keeps track of how many times it has been scheduled and what it did when it was scheduled, then the analogue of $P^2(k, m)$ implements both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$ if messages are guaranteed to arrive (where now $P^2(k, m)$ means that if $bit = 0$, then the kth time that S is scheduled it performs sendbit, while if $bit = 1$, then the mth time that S is scheduled it performs sendbit). Similarly, the analogue of P^ω implements both $\mathsf{BT}^{>}$ and $\mathsf{BT}^{\Diamond B}$ in contexts that satisfy the fairness assumption (but any finite number of messages may not arrive).

4 Discussion

This paper presents a framework that facilitates high-level counterfactual reasoning about protocols. Indeed, it enables the design of well-defined protocols in which processes act based on their knowledge of counterfactual statements. This is of interest because, in many instances, the intuition behind the choice of a given course of action is best thought of and described in terms of counterfactual reasoning. For example, it is sometimes most efficient for agents to stop exending resources once they know that their goals will be achieved even if they stop. Making this precise involves counterfactual reasoning; this agent must consider what would happen were it to stop expending resources.

This paper should perhaps best be viewed as a "proof of concept"; the examples involving the bit-transmission program show that counterfactuals can play a useful role in knowledge-based programs. While we have used standard approaches to giving semantics to belief and counterfactuals (adapated to the runs and systems framework that we are using), these definitions give the user a large number of degrees of freedom, in terms of choosing the ranking function to define belief and the notion of closeness needed to define counterfactuals. While we have tried to suggest some reasonable choices for how the ranking function and the notion of closeness are defined, and these choices certainly gave answers that matched our intuitions in all the context we considered for the bit-transmission problem, it would be helpful to have a few more examples to test the reasonableness of the choices. We are currently exploring the application of cbb programs for analyzing message-efficient leader election in various topologies; we hope to report on this in future work.

While we used the very simple problem of bit transmission as a vehicle for introducing our framework for knowledge, belief, and counterfactuals, we believe it should be useful for handling a much broader class of distributed protocols. We gave an example of how counterfactual reasoning is useful in deciding whether a message needs to be sent. Similar issues arise, for example, in deciding whether to perform a write action on a shared-memory variable. Because our framework provides a concrete model for understanding the interaction between belief and counterfactuals, and for defining the notion of "closeness" needed for interpreting counterfactuals, it should also be useful for illuminating some problems in philosophy and game theory. The insight our analysis gave to the procrastinator's paradox is an example of how counterfactual programs can be related to issues in the philosophy of human behavior. We believe that, in particular, the framework will be helpful in understanding some extensions of Nash equilibrium in game theory. For example, as we saw in Lemma 3.7, whether a protocol de facto implements a cbb program depends on the agent's beliefs. This seems closely related to the notion of a *subjective equilibrium* in game theory [Kalai and Lehrer 1995]. We are currently working on drawing a formal connection between our framework notions of equilibrium in game theory. It would also be interesting to relate the notion of "closeness" defined in our framework to that given by the structural-equations model used by Pearl [2000] (see also [Halpern 2000]). The structural-equations model also gives a concrete interpretation to "closeness"; it does so in terms of mechanisms defined by equations. It would be interesting to see if these mechanisms can be modeled as protocols in a way that makes the definitions agree.

References

Dwork, C. and Y. Moses (1990). Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation 88*(2), 156–186.

Engelhardt, K., R. van der Meyden, and Y. Moses (1998). Knowledge and the logic of local propositions. In *Theoretical Aspects of Rationality and Knowledge: Proc. Seventh Conference (TARK 1998)*, pp. 29–41.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1995). *Reasoning about Knowledge*. Cambridge, Mass.: MIT Press.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1997). Knowledge-based programs. *Distributed Computing 10*(4), 199–225.

Francez, N. (1986). *Fairness*. Berlin/New York: Springer-Verlag.

Friedman, N. and J. Y. Halpern (1997). Modeling belief in dynamic systems. Part I: foundations. *Artificial Intelligence 95*(2), 257–316.

Goldszmidt, M. and J. Pearl (1992). Rank-based systems: A simple approach to belief revision, belief update and reasoning about evidence and actions. In *Principles of Knowledge Representation and Reasoning: Proc. Third International Conference (KR '92)*, pp. 661–672.

Hadzilacos, V. (1987). A knowledge-theoretic analysis of atomic commitment protocols. In *Proc. 6th ACM Symp. on Principles of Database Systems*, pp. 129–134.

Hadzilacos, V. and J. Y. Halpern (1993). Message-optimal protocols for Byzantine agreement. *Mathematical Systems Theory 26*, 41–102.

Halpern, J. Y. (2000). Axiomatizing causal reasoning. *Journal of A.I. Research 12*, 317–337.

Halpern, J. Y. and R. Fagin (1989). Modelling knowledge and action in distributed systems. *Distributed Computing 3*(4), 159–179. A preliminary version appeared in *Proc. 4th ACM Symposium on Principles of Distributed Computing*, 1985, with the title "A formal model of knowledge, action, and communication in distributed systems: preliminary report".

Halpern, J. Y. and Y. Moses (1990). Knowledge and common knowledge in a distributed environment. *Journal of the ACM 37*(3), 549–587.

Halpern, J. Y., Y. Moses, and O. Waarts (2001). A characterization of eventual Byzantine agreement. *SIAM Journal on Computing 31*(3), 838–865.

Halpern, J. Y. and L. D. Zuck (1992). A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM 39*(3), 449–478.

Kalai, E. and E. Lehrer (1995). Subjective games and equilibria. *Games and Economic Behavior 8*, 123–163.

Lewis, D. K. (1973). *Counterfactuals.* Cambridge, Mass.: Harvard University Press.

Manna, Z. and A. Pnueli (1992). *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Berlin/New York: Springer-Verlag.

Mazer, M. S. (1990). A link between knowledge and communication in faulty distributed systems. In *Theoretical Aspects of Reasoning about Knowledge: Proc. Third Conference*, pp. 289–304.

Mazer, M. S. and F. H. Lochovsky (1990). Analyzing distributed commitment by reasoning about knowledge. Technical Report CRL 90/10, DEC-CRL.

Moses, Y. and M. R. Tuttle (1988). Programming simultaneous actions using common knowledge. *Algorithmica 3*, 121–169.

Neiger, G. and S. Toueg (1993). Simulating real-time clocks and common knowledge in distributed systems. *Journal of the ACM 40*(2), 334–367.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference.* New York: Cambridge University Press.

Spohn, W. (1988). Ordinal conditional functions: a dynamic theory of epistemic states. In W. Harper and B. Skyrms (Eds.), *Causation in Decision, Belief Change, and Statistics*, Volume 2, pp. 105–134. Dordrecht, Netherlands: Reidel.

Stalnaker, R. C. (1968). A theory of conditionals. In N. Rescher (Ed.), *Studies in Logical Theory*, American Philosophical Quarterly Monograph Series, No. 2, pp. 98–112. Oxford, U.K.: Blackwell. Also appears in W. L. Harper, R. C. Stalnaker and G. Pearce (Eds.), *Ifs.* Dordrecht, Netherlands: Reidel, 1981.

Stalnaker, R. C. (1980). A defense of conditional excluded middle. In W. L. Harper, R. Stalnaker, and G. Pearce (Eds.), *Ifs*, pp. 87–104. Dordrecht, Netherlands: Reidel.